(12) UK Patent Application (19) GB (11) 2 169 114 A

(43) Application published 2 Jul 1986

(21) Application No 8530185

(22) Date of filing 12 Nov 1984

Date lodged 6 Dec 1985

(30) Priority data
(31) 550693 (32) 10 Nov 1983 (33) US

(60) Derived from Application No 8428580 under Section 15(4) of the Patents Act 1977

(51) INT CL⁴
G06F 7/48

(52) Domestic classification (Edition H):
G4A CX

(56) Documents cited
EP A2 0112676 EP A1 0036605 EP A2 0096165
EP A1 0108655

(58) Field of search
G4A
H4P
Selected US specifications from IPC sub-class G06F

(71) Applicant
General Signal Corporation (USA–New York), High Ridge Park, Stamford, Connecticut 06904, United States of America

(72) Inventor
David B Rutherford

(74) Agent and/or Address for Service
W P Keppler, Central Patent Department (Wembley Office), The General Electric Company p l c, Hirst Research Centre, Wembley, Middlesex HA9 7PP

(54) Vital processor V

(57) A multi-bit binary value is computed from at least two input multi-bit binary values using a feedback shift register which can be arranged to provide maximal length sequences. The shift register is preconditioned for evaluation in one of two data channels by inserting predetermined bit vlaues into its stages Q_0–Q_{31}, and then performing a given number of shifts with either of the channel specific feedback schemes PE1 or PE2 operative. Subsequently, the first input multi-bit binary value is added to the channel specific bits in the shift register, via inputs D0–D31, whence the register is cyclically shifted a predetermined number of places without internal feedback. Cycles of binary value entry and shift occur until all input multi-bit binary values are used, whence the register contains the computed value. The shift register may be used in a feedback mode (Fig. 10) to confirm that each bit pattern comprises a data field and a check field related by a predetermined code rule.

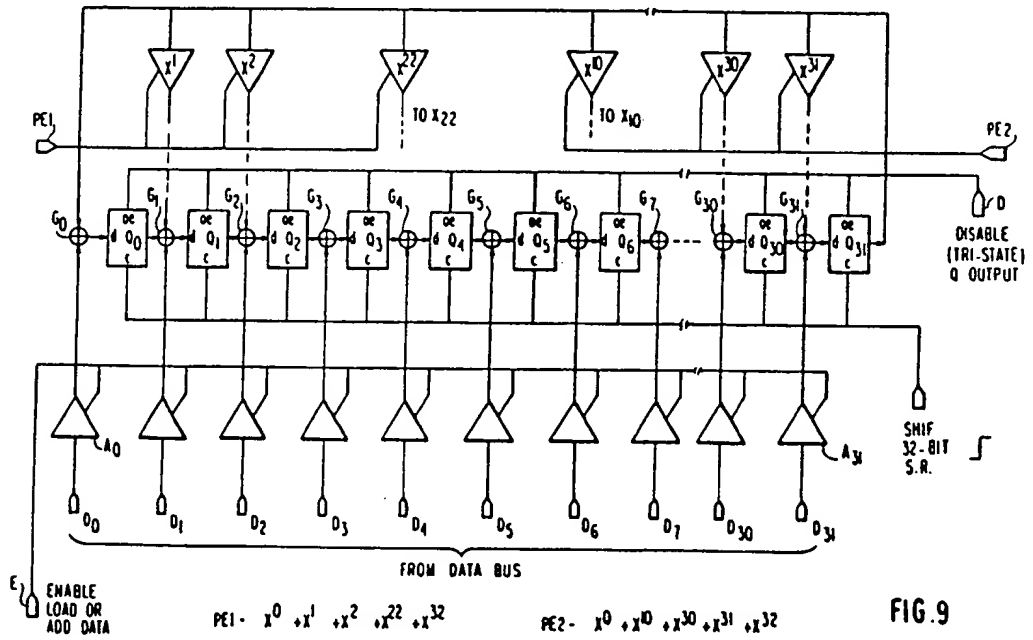
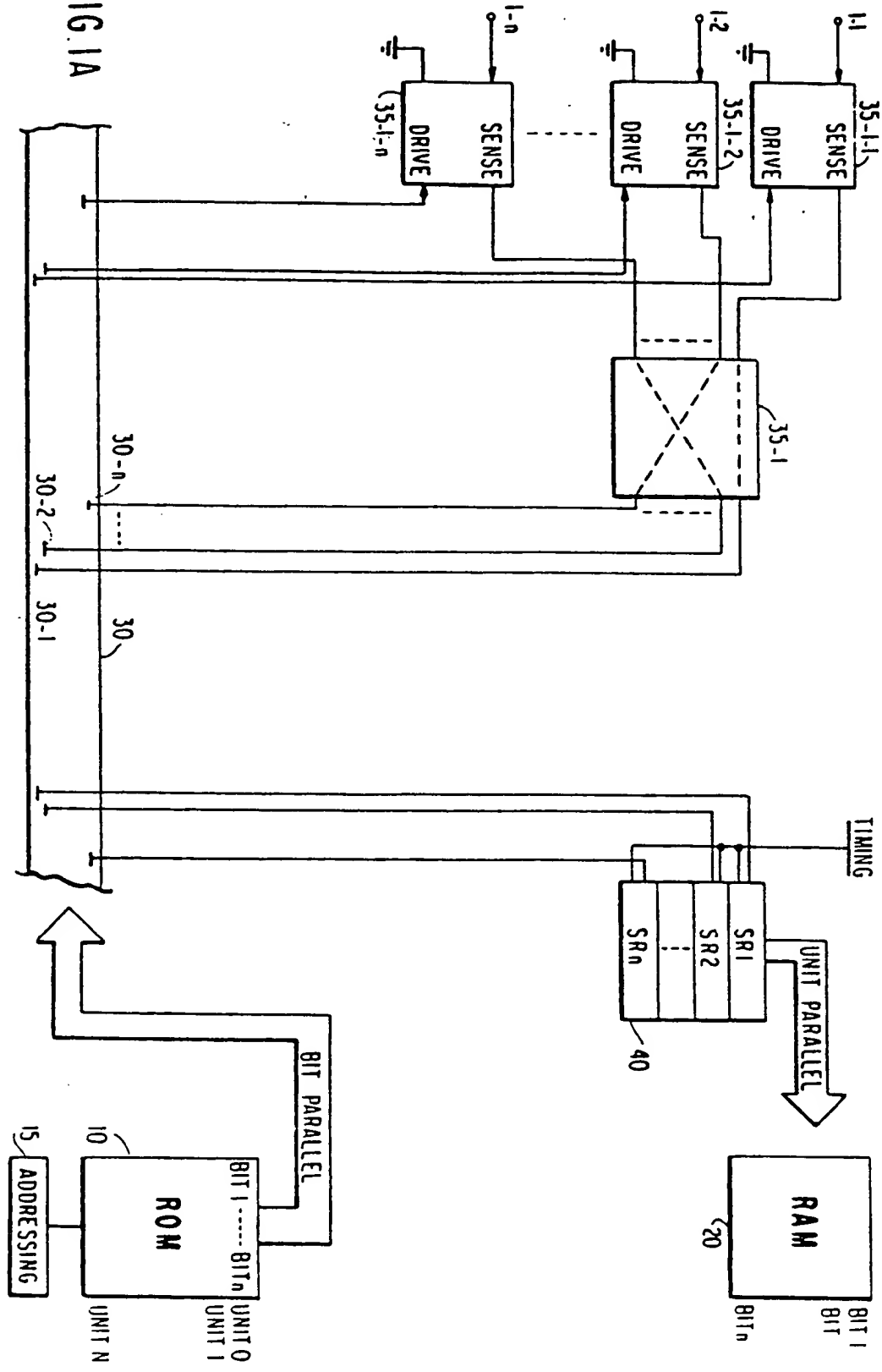


FIG.9

GB 2 169 114 A

FIG. 1A


I-1
I-2
I-n
35-1-1
35-1-2
35-1-n
DRIVE
SENSE
35-1
30
30-1
30-2
30-n
TIMING
SR1
SR2
SRn
40
UNIT PARALLEL
BIT PARALLEL
RAM
20
BIT 1
BIT n
ROM
10
15
ADDRESSING
UNIT 0
UNIT 1
UNIT N

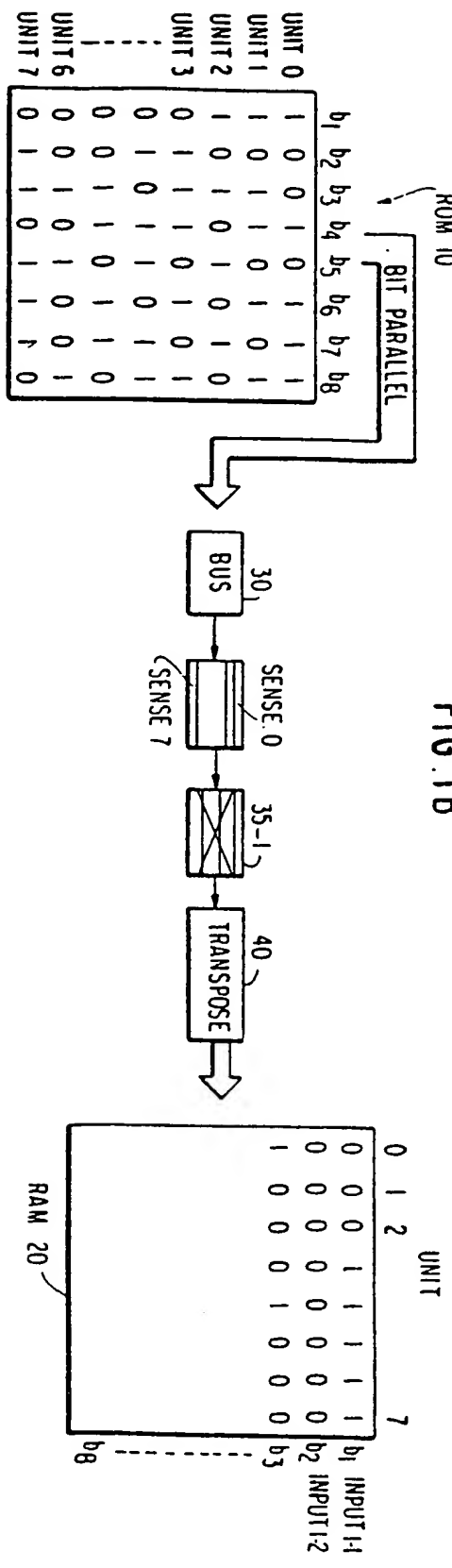

ROM 10
BIT PARALLEL
UNIT 0
UNIT 1
UNIT 2
UNIT 3
UNIT 6
UNIT 7
b1 b2 b3 b4 b5 b6 b7 b8
BUS
30
SENSE 7
SENSE 0
35-1
TRANSPOSE
40
RAM 20
UNIT
0 1 2 7
INPUT 1-1
INPUT 1-2

FIG. 1B

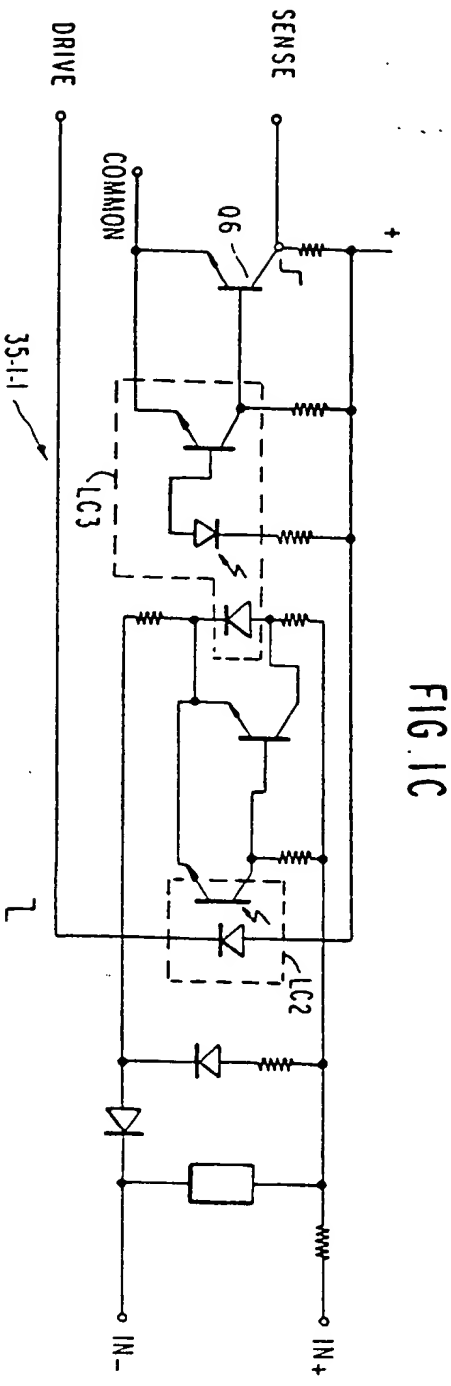

DRIVE
SENSE
COMMON
Q6
35-1-1
LC3
LC2
L
IN-
IN+

FIG. 1C

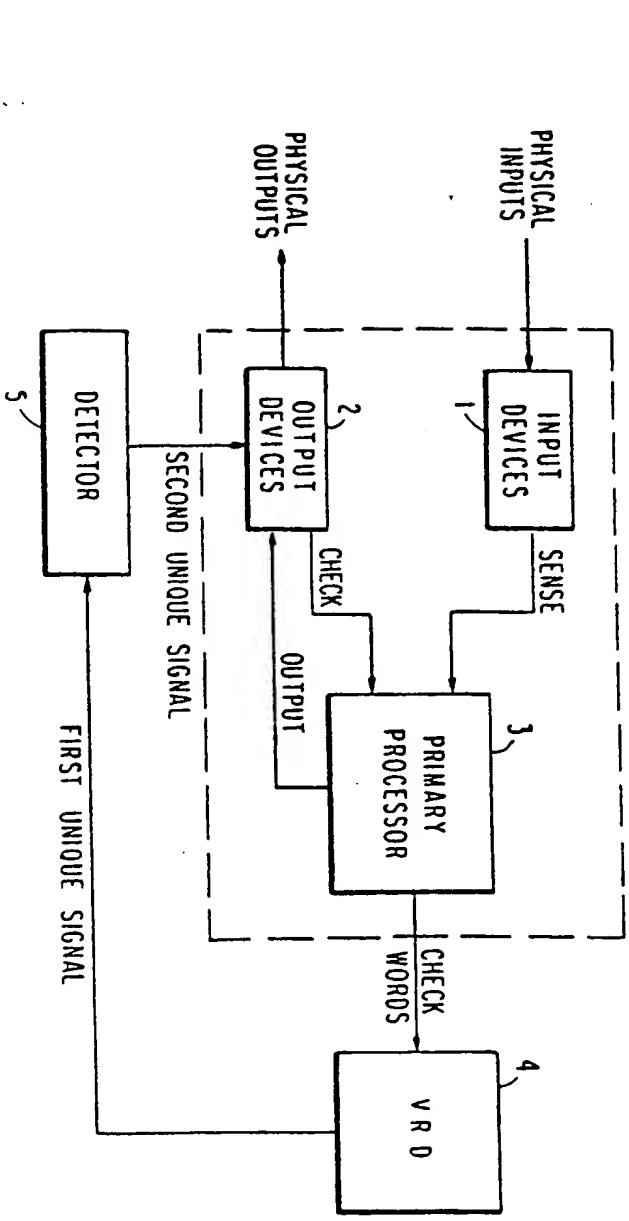

PHYSICAL INPUTS
PHYSICAL OUTPUTS
INPUT DEVICES
1
OUTPUT DEVICES
2
SENSE
CHECK
OUTPUT
PRIMARY PROCESSOR
3
CHECK WORDS
VRD
4
DETECTOR
5
SECOND UNIQUE SIGNAL
FIRST UNIQUE SIGNAL

FIG. 1D

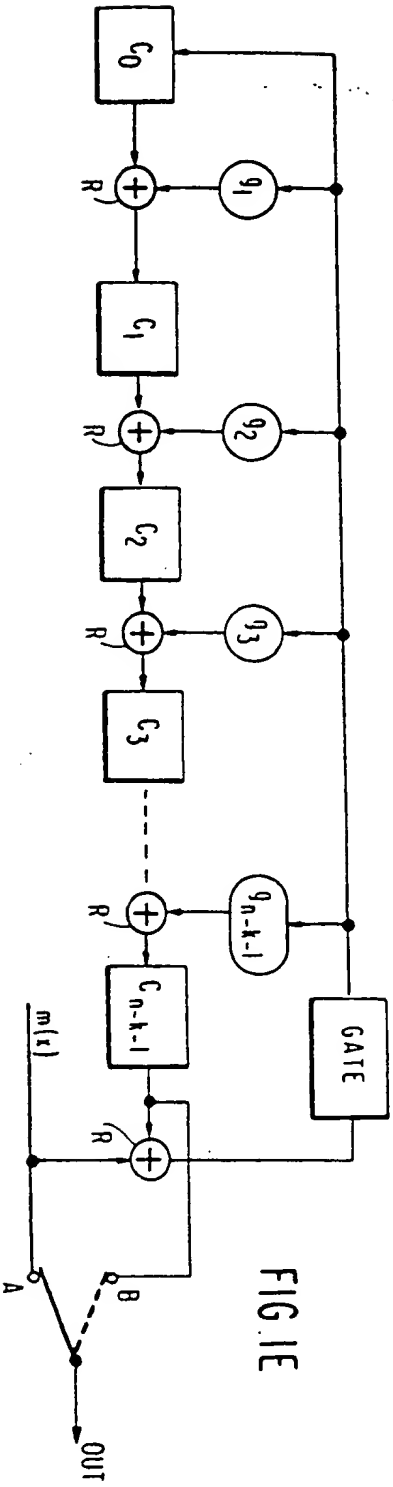

C_0
C_1
C_2
C_3
C_{n-k-1}
g_1
g_2
g_3
g_{n-k-1}
R
GATE
m(x)
A
B
OUT

FIG. 1E

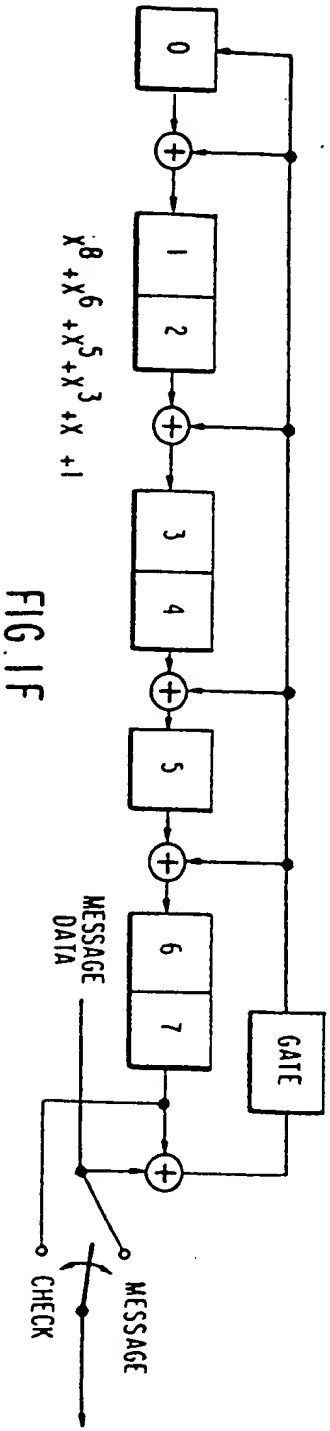

0
1
2
3
4
5
6
7
GATE
MESSAGE DATA
CHECK
MESSAGE
$X^8+X^6+X^5+X^3+X+1$

FIG. 1F

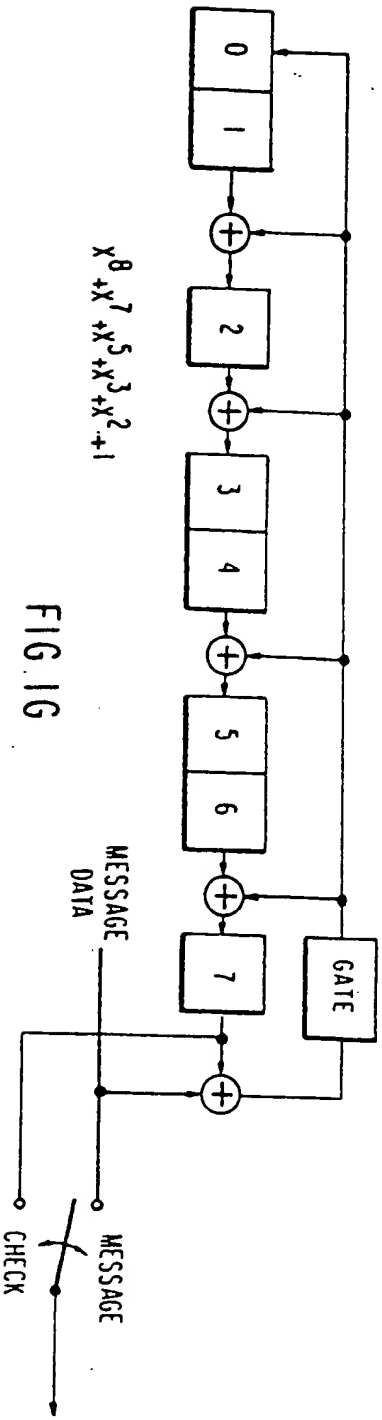

0
1
2
3
4
5
6
7
GATE
MESSAGE DATA
CHECK
MESSAGE
$X^8+X^7+X^5+X^3+X^2+1$

FIG. 1G

FIG. 1H

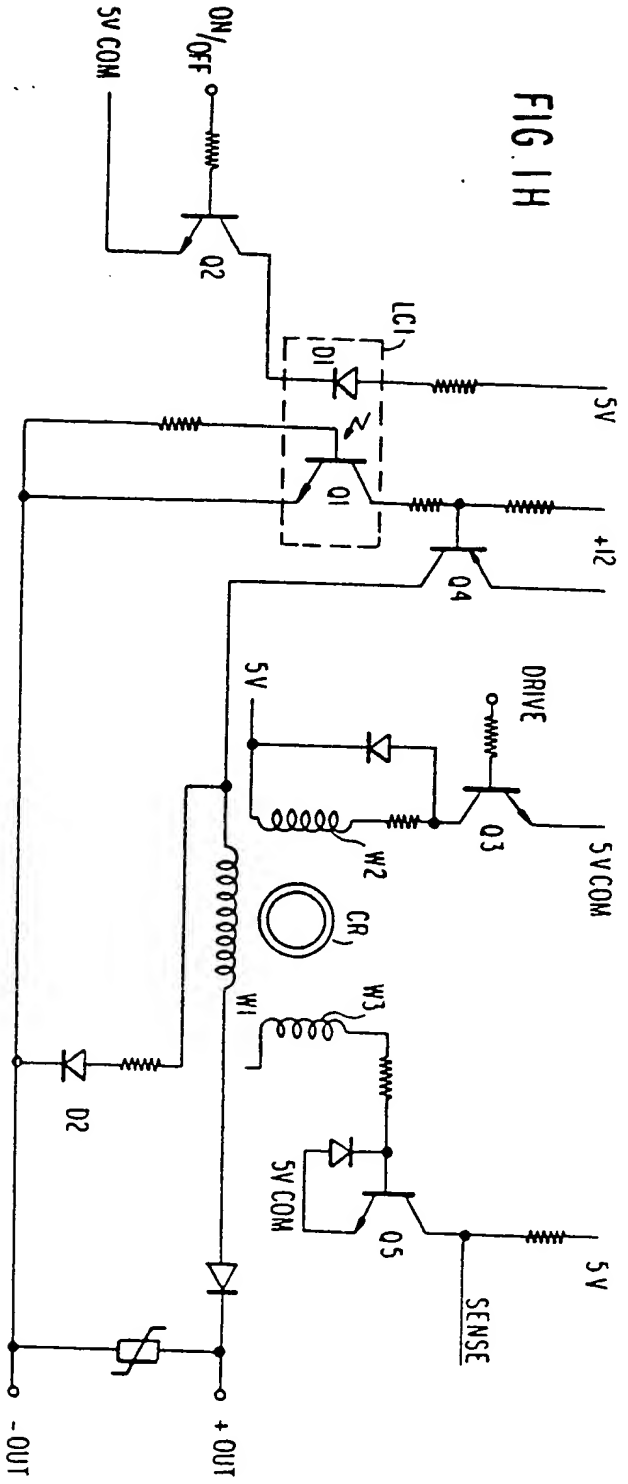
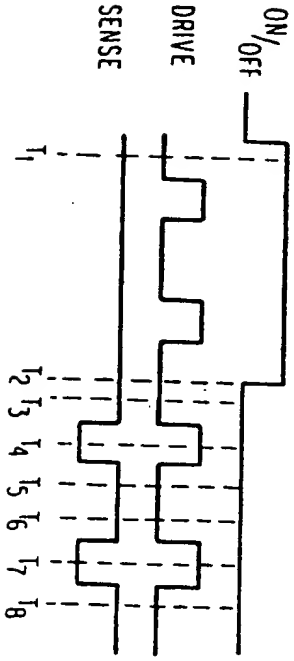

ON/OFF
5V COM
Q2
LC1
D1
5V
Q1
+12
Q4
5V
DRIVE
Q3
5V COM
W2
CR1
W1
W3
D2
5V COM
Q5
5V
SENSE
- OUT
+ OUT

FIG. 1J


ON/OFF
DRIVE
SENSE
T1
T2 T3 T4 T5 T6 T7 T8

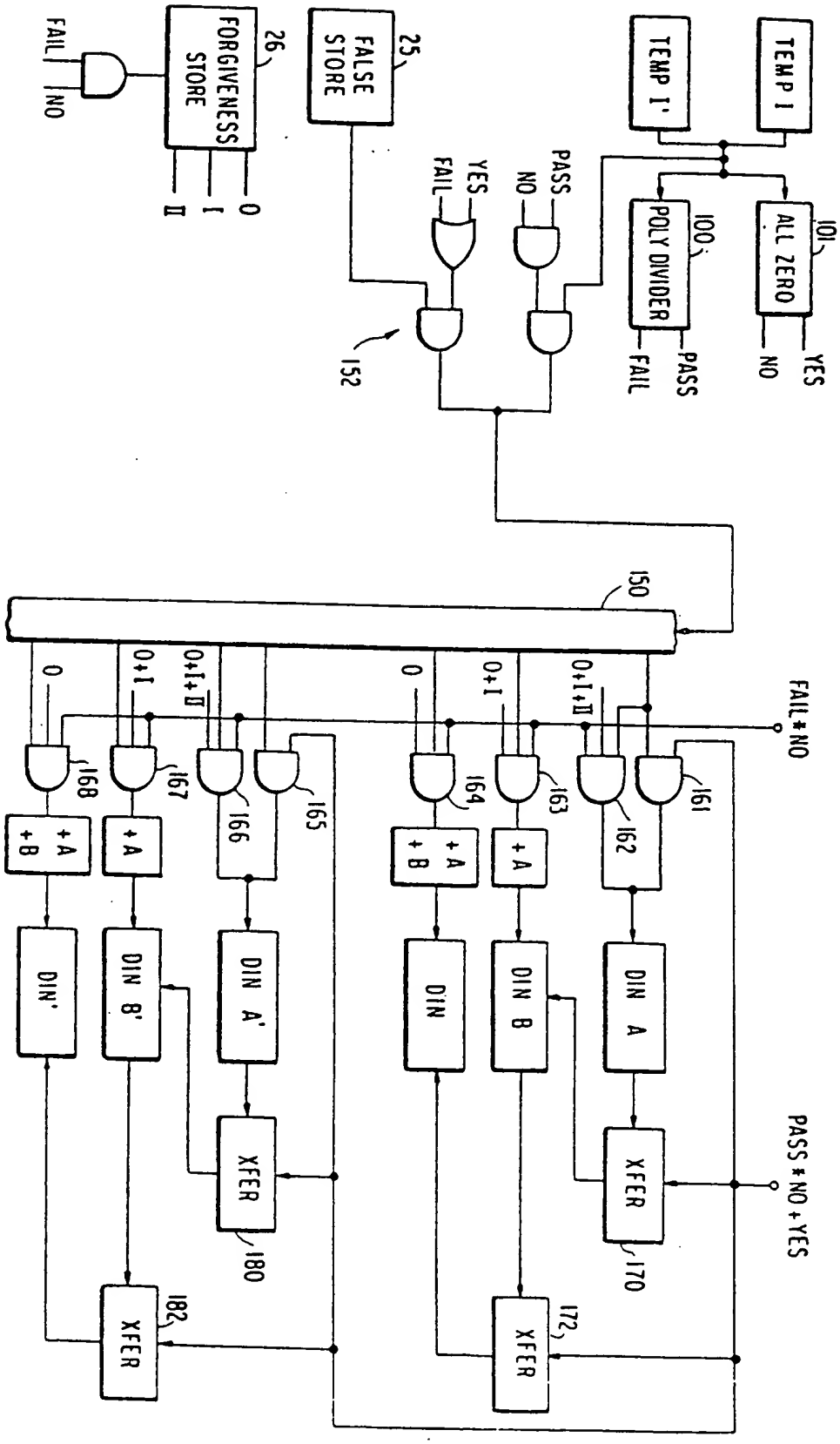
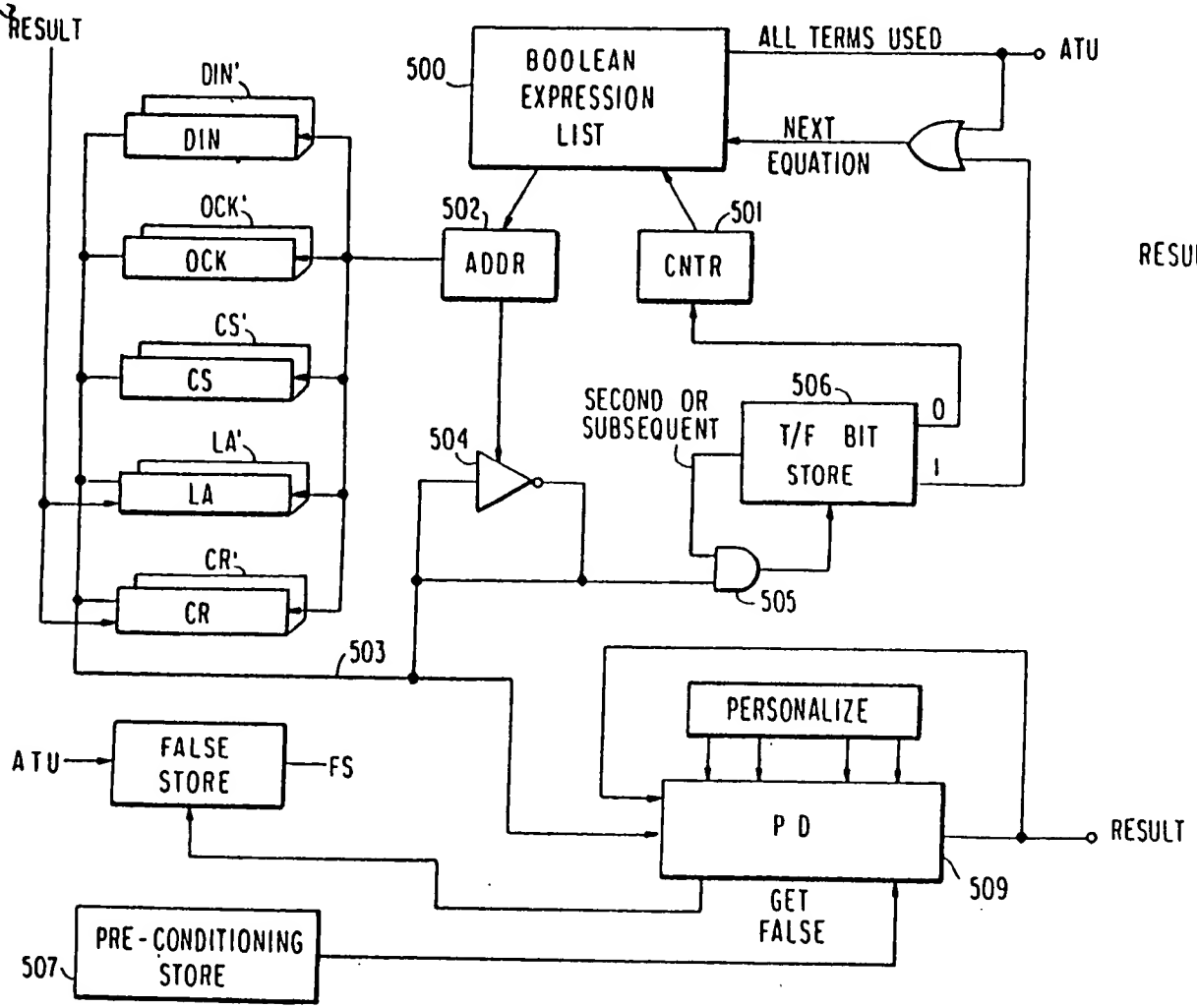
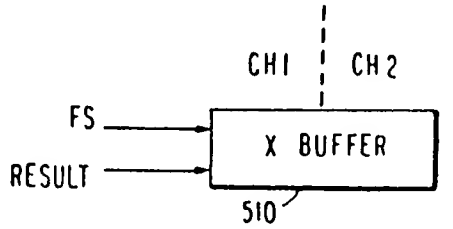

TEMP I
TEMP I'
ALL ZERO
YES
NO
101
POLY DIVIDER
PASS
FAIL
100
PASS
NO
YES
FAIL
152
FALSE STORE
25
FORGIVENESS STORE
26
FAIL
NO
0
I
II
150
FAIL*NO
PASS*NO+YES
0
0+I
0+I+II
DIN
DIN A
DIN B
DIN'
DIN A'
DIN B'
+A
+A +B
XFER
161
162
163
164
165
166
167
168
170
172
180
182

FIG. 2





$$OF_1 = \underbrace{IF_1 \cdot IF_3}_{\text{TERM 1}} + \underbrace{IF_2 \cdot \overline{IF}_4}_{\text{TERM 2}} + \cdots$$

FIG. 3

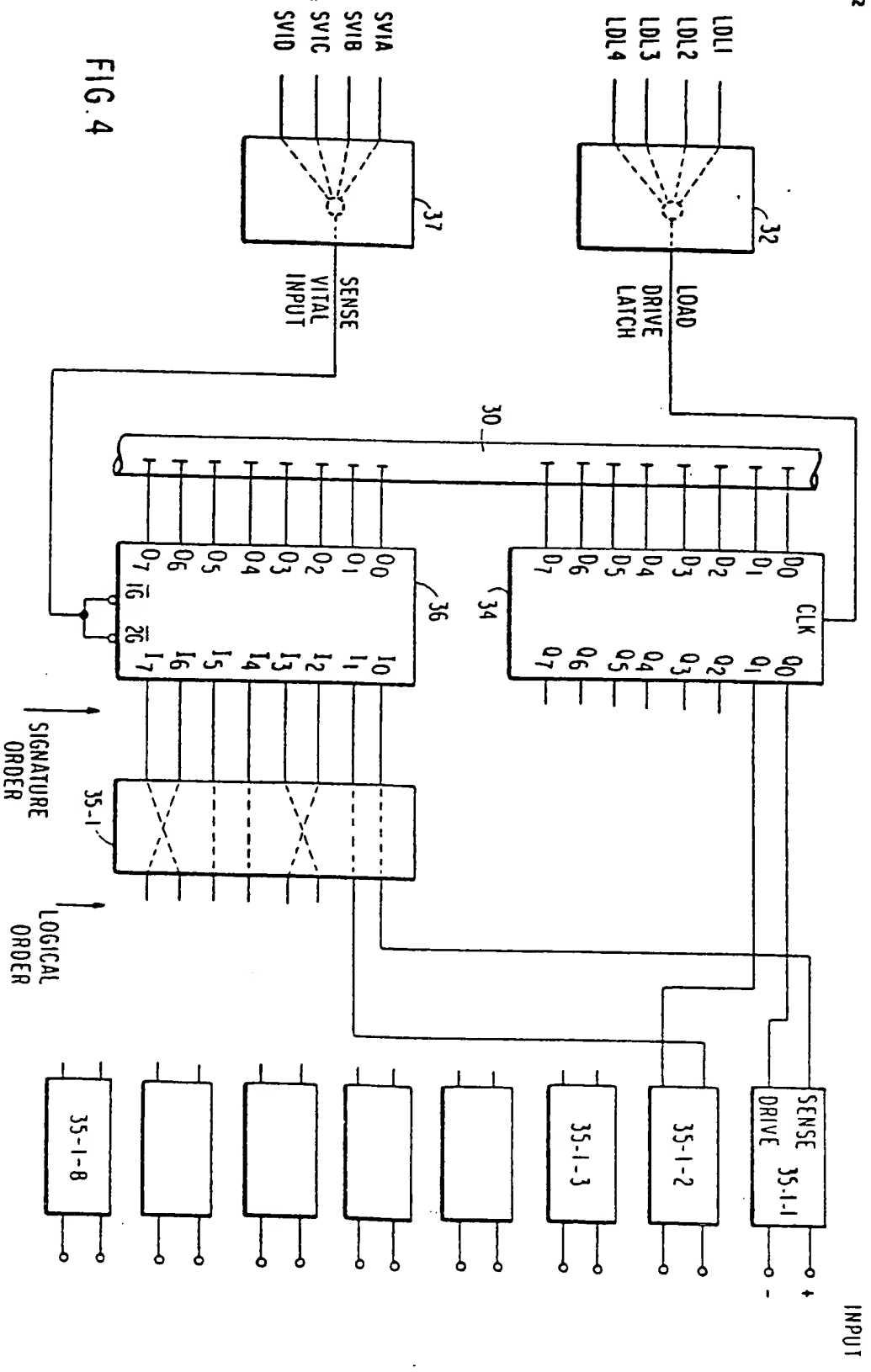

LDL1
LDL2
LDL3
LDL4
32
LOAD DRIVE LATCH
SVIA
SVIB
SVIC
SVID
37
SENSE VITAL INPUT
30
34
36
CLK
SIGNATURE ORDER
35-1
LOGICAL ORDER
35-1-8
35-1-3
35-1-2
SENSE DRIVE 35-1-1
INPUT

FIG. 4

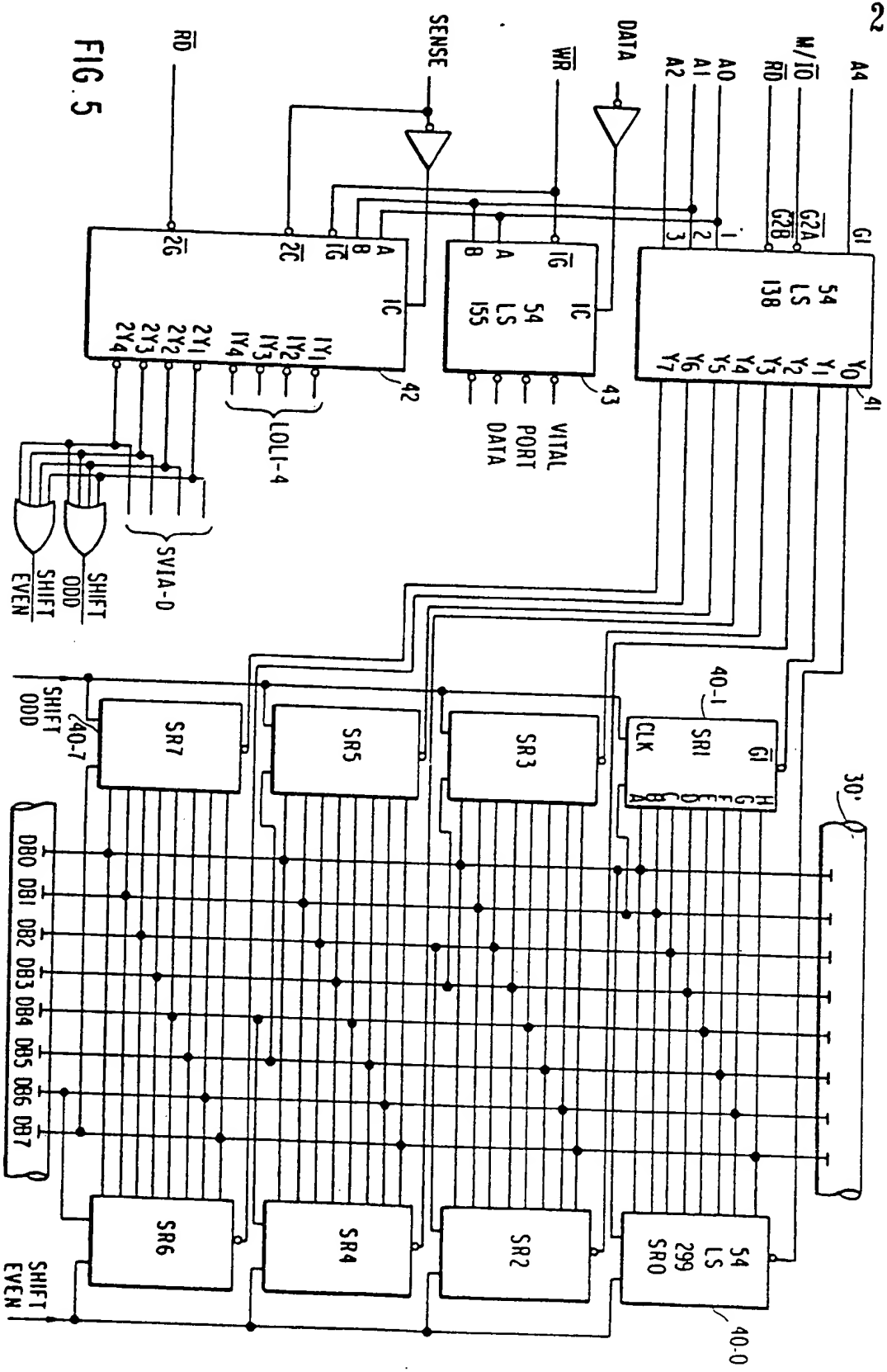

A4
M/IO
RD
A0
A1
A2
DATA
WR
SENSE
RD
G1
G2A
G2B
54 LS 138
Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7
41
54 LS 155
43
VITAL PORT DATA
42
LOLI-4
SVIA-0
SHIFT ODD
SHIFT EVEN
40-1
SR1
CLK
40-7
SR7
SR5
SR3
30'
DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7
SR6
SR4
SR2
54 LS 299
SR0
40-0

FIG. 5

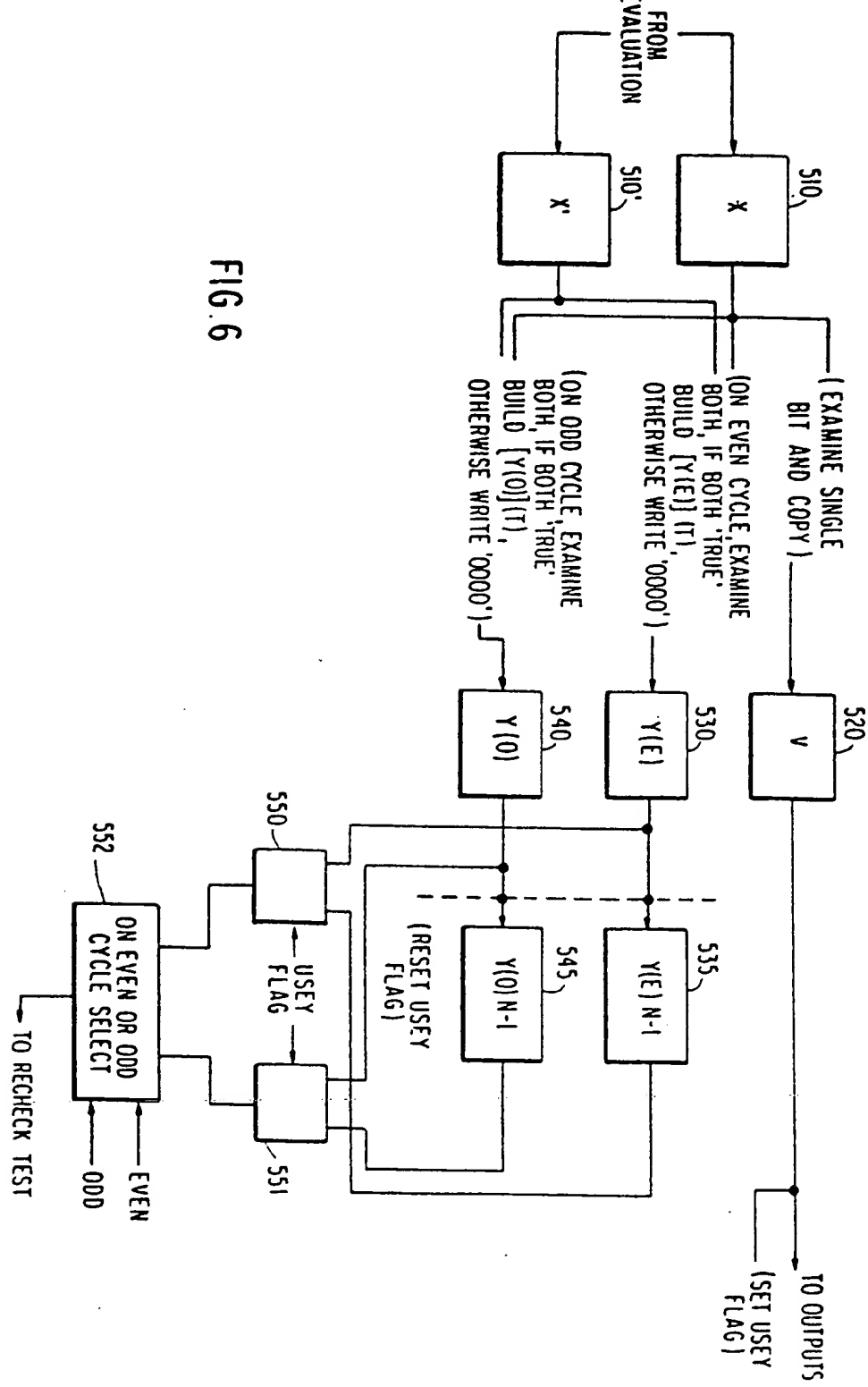

FROM EVALUATION
X'
510'
X
510
(EXAMINE SINGLE BIT AND COPY)
(ON EVEN CYCLE, EXAMINE BOTH, IF BOTH 'TRUE' BUILD [Y(E)](T), OTHERWISE WRITE '0000')
(ON ODD CYCLE, EXAMINE BOTH, IF BOTH 'TRUE' BUILD [Y(O)](T), OTHERWISE WRITE '0000')
V
520
Y(E)
530
Y(O)
540
Y(E) N-1
535
Y(O) N-1
545
(RESET USEY FLAG)
550
USEY FLAG
551
552
ON EVEN OR ODD CYCLE SELECT
EVEN
ODD
TO RECHECK TEST
TO OUTPUTS
(SET USEY FLAG)

FIG.6

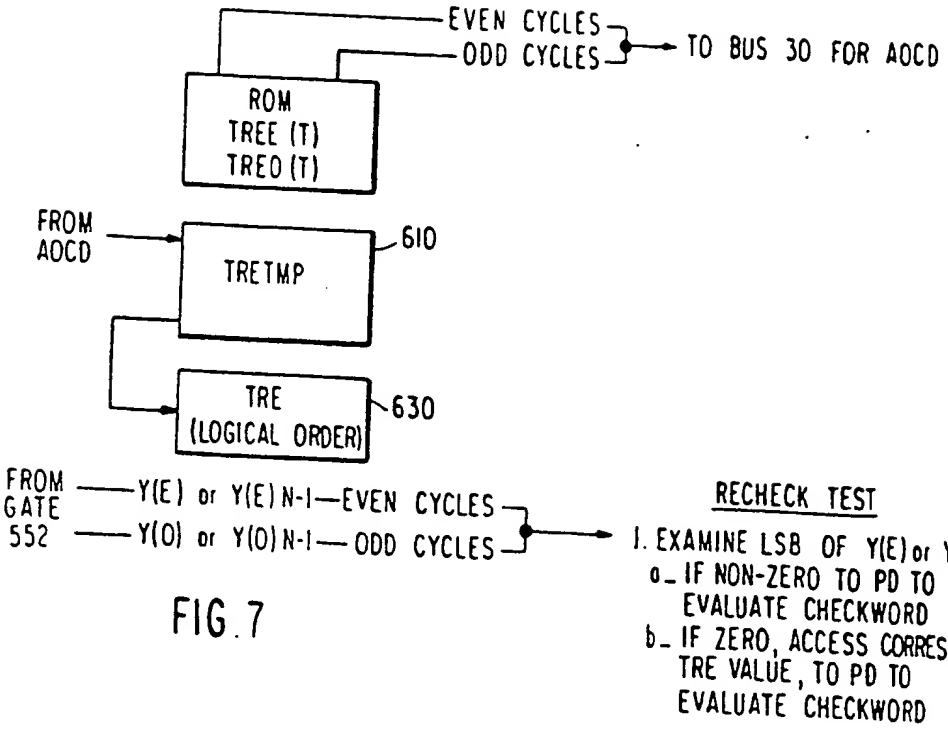
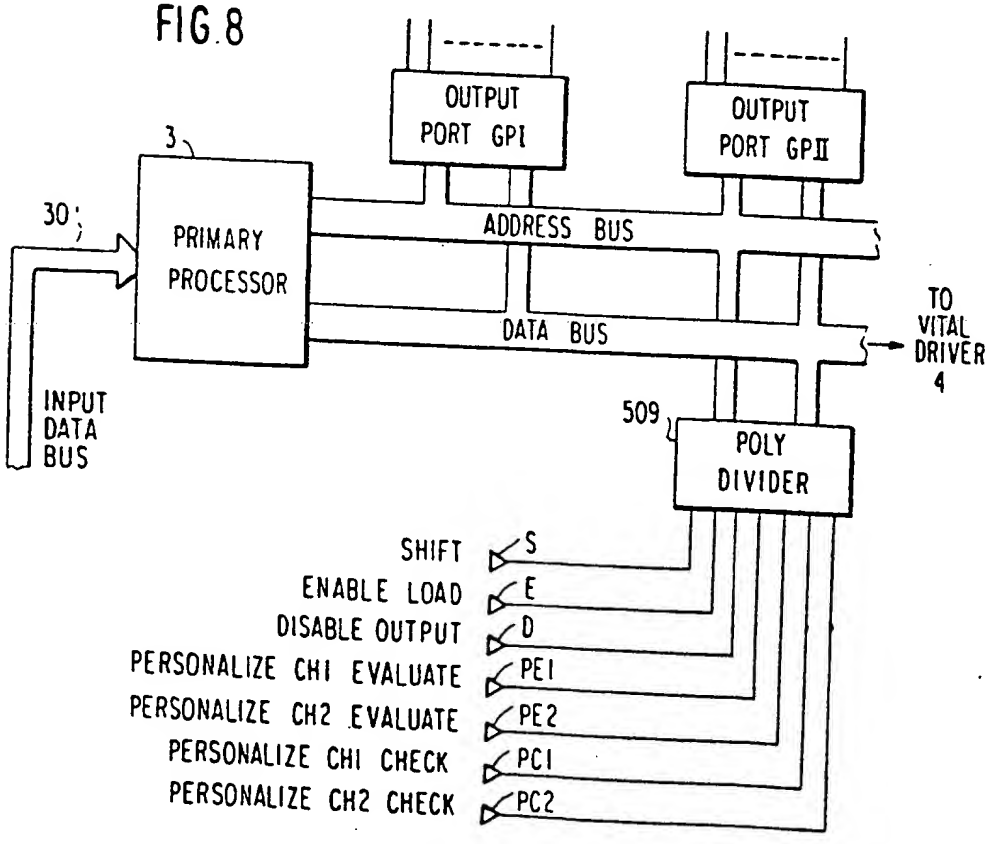
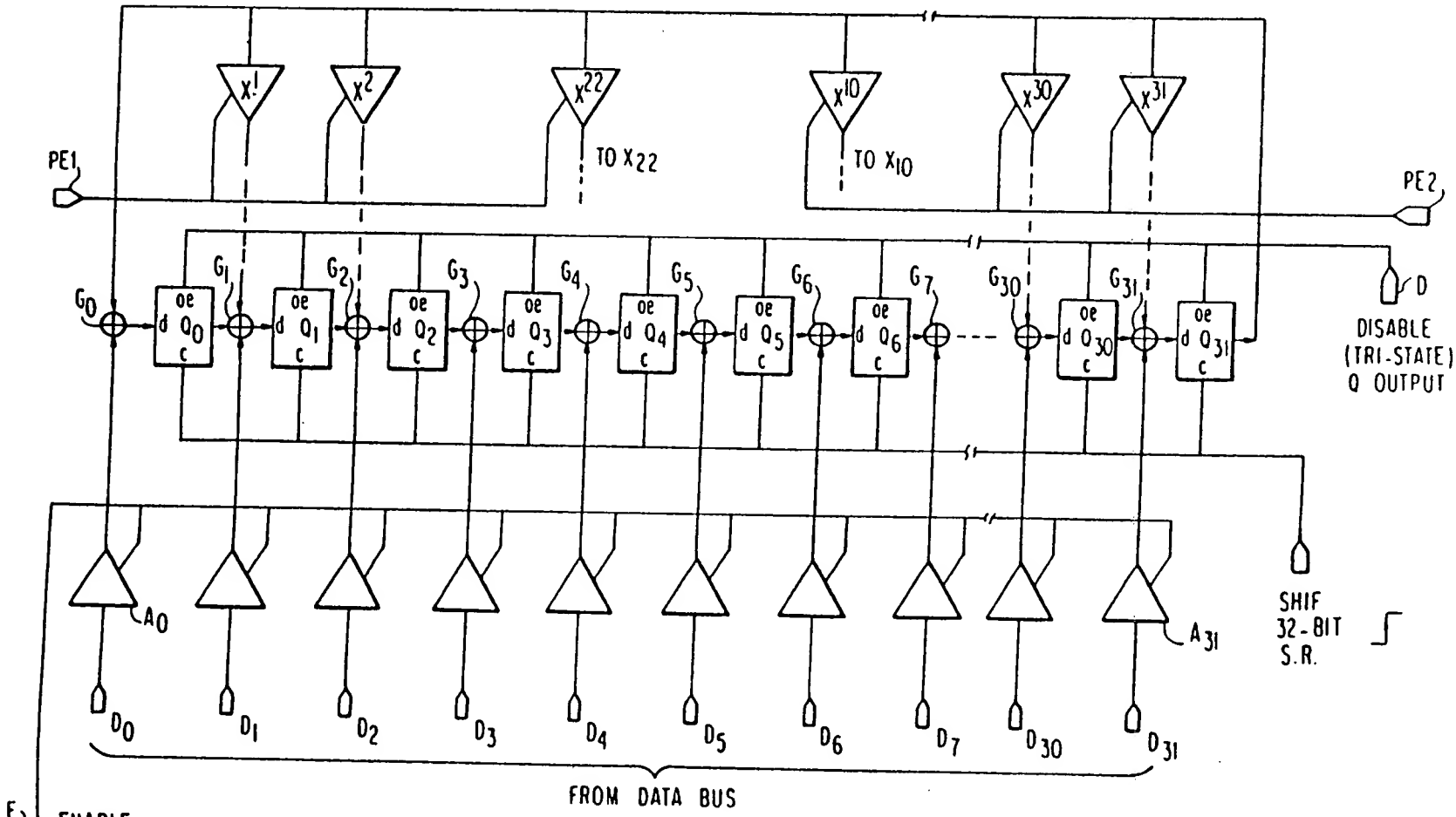

EVEN CYCLES
ODD CYCLES
TO BUS 30 FOR AOCD
ROM
TREE (T)
TREO (T)
FROM
AOCD
TRETMP
610
TRE
(LOGICAL ORDER)
630
FROM
GATE
552
Y(E) or Y(E) N-1—EVEN CYCLES
Y(O) or Y(O) N-1— ODD CYCLES
RECHECK TEST
1. EXAMINE LSB OF Y(E) or Y(O)
a- IF NON-ZERO TO PD TO
EVALUATE CHECKWORD
b- IF ZERO, ACCESS CORRESPONDING
TRE VALUE, TO PD TO
EVALUATE CHECKWORD

FIG. 7

FIG. 8


OUTPUT
PORT GPI
OUTPUT
PORT GPII
3
30
PRIMARY
PROCESSOR
ADDRESS BUS
DATA BUS
TO
VITAL
DRIVER
4
INPUT
DATA
BUS
509
POLY
DIVIDER
SHIFT
S
ENABLE LOAD
E
DISABLE OUTPUT
D
PERSONALIZE CH1 EVALUATE
PE1
PERSONALIZE CH2 EVALUATE
PE2
PERSONALIZE CH1 CHECK
PC1
PERSONALIZE CH2 CHECK
PC2



PE1 - $X^0 + X^1 + X^2 + X^{22} + X^{32}$

PE2 - $X^0 + X^{10} + X^{30} + X^{31} + X^{32}$

FIG. 9

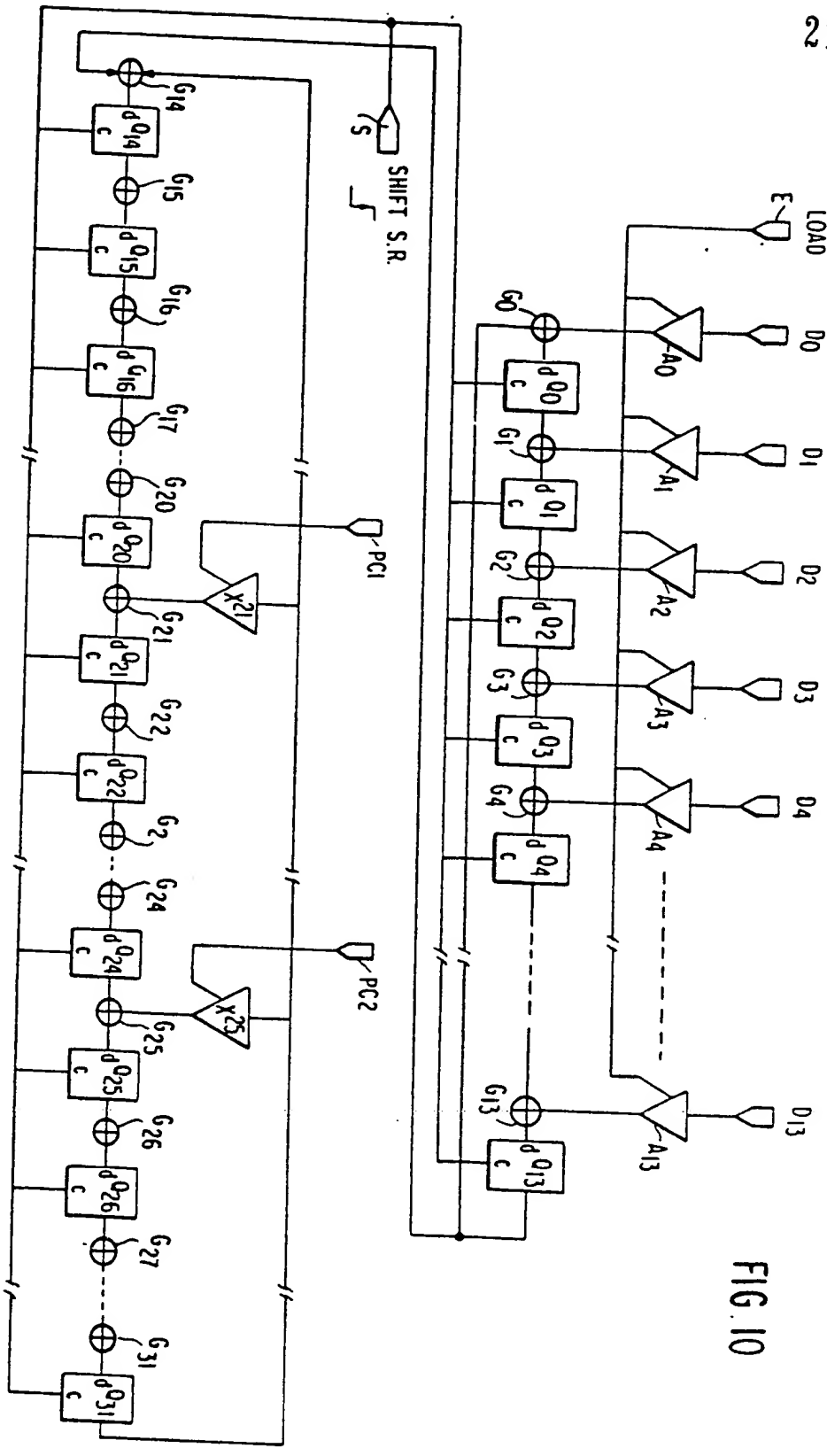


FIG. 10

FIG.11

DINADS HEAD BLOCK (DIHEAD)

OFFSET	CONTENTS
0	# DIRECT INPUT PORTS (USED AND UNUSED)
2	# INPUT PORT GROUPS (1 GROUP = 16 PORTS)
4	CH2 RAM OFFSET (OFFSET FROM CHIBPT)
6	<DIN> BUFFER SIZE (IN WORDS)
8	<DIN>START ADDRESS (OFFSET FROM RAMSTART)
A	<DINA> START ADDRESS (OFFSET FROM RAMSTART)
C	<DINB>START ADDRESS (OFFSET FROM RAMSTART)
E	<TEMPI> START ADDRESS (OFFSET FROM RAMSTART)
10	KIO (LO)
12	KIO (HI)
14	KIO' (LO)
16	KIO' (HI)
18	KIA (LO)
1A	KIA (HI)
1C	KIA' (LO)
1E	KIA' (HI)
20	KIB (LO)
22	KIB (HI)
24	KIB' (LO)
26	KIB' (HI)
28	KIT (LO)
2A	KIT (HI)
2C	KIT' (LO)
2E	KIT' (HI)
30	PREIN (LO)
32	PREIN (HI)
34	PREIA (LO)
36	PREIA (HI)
38	PREIB (LO)
3A	PREIB (HI)
3C	PREIT (LO)
3E	PREIT (HI)
40	CKIADR ADDR OF 1st SLOT IN <W> (MAIN) FOR 4 CHECKWORDS
42	CKTM12 CHECKWORD ADDRESS IN <W> (MAIN)
44	CKTM23 CHECKWORD ADDRESS IN <W> (MAIN)
46	CKTM34 CHECKWORD ADDRESS IN <W> (MAIN)

DIGRXY

FIG.12

OFFSET	CONTENTS
0	POINTER TO NEXT DIGRXY HEAD BLOCK
2	INPUT PORT GROUP X,Y BOARD ADDRESS
4	# OF PORTS IN GROUP X,Y
6	2nd SUB-GROUP PORT TYPE | 1st SUB-GROUP PORT TYPE
8	PTR TO TOP OF SECTION X,Y IN <TEMPI>
A	PTR TO TOP OF SECTION X,Y IN <DIN>
C	PTR TO TOP OF SECTION X,Y IN <DINA>
E	PTR TO TOP OF SECTION X,Y IN <DINB>
10	PTR TO TOP OF DIS1XY DATA BLOCK
12	PTR TO TOP OF DIS2XY DATA BLOCK
14	PTR TO TOP OF TIL1XY DATA BLOCK
16	PTR TO TOP OF TIL2XY DATA BLOCK

FIG.13

DIS1XY

OFFSET	CONTENTS
0	(S0,X,Y) PORT TYPE | LOGICAL PORT # | # CYCLES OF FORGIVENESS | SIG. ORDER SR # |
2	DI (S0,X,Y)(F)(LO)
4	DI (S0,X,Y)(F)(HI)
6	(S1,X,Y) PORT TYPE | LOGICAL PORT # | # CYCLES OF FORGIVENESS | SIG. ORDER SR # |
8	DI I (S1,X,Y)(F)(LO)
A	DI I (S1,X,Y)(F)(HI)
C	(S2,X,Y) PORT TYPE | LOGICAL PORT # | # CYCLES OF FORGIVENESS | SIG. ORDER SR # |
E	DI I (S2,X,Y)(F)(LO)
10	DI I (S2,X,Y)(F)(HI)
12	(S3,X,Y) PORT TYPE | LOGICAL PORT # | # CYCLES OF FORGIVENESS | SIG. ORDER SR # |
14	DI I (S3,X,Y)(F)(LO)
16	DI I (S3,X,Y)(F)(HI)
18	(S4,X,Y) PORT TYPE | LOGICAL PORT # | # CYCLES OF FORGIVENESS | SIG. ORDER SR # |
1A	DI I (S4,X,Y)(F)(LO)
1C	DI I (S4,X,Y)(F)(HI)
1E	(S5,X,Y) PORT TYPE | LOGICAL PORT # | # CYCLES OF FORGIVENESS | SIG. ORDER SR # |
	...
5C	DI I (SF,X,Y)(F)(LO)
5E	DI I (SF,X,Y)(F)(HI)

FIG 14

TILI(L,X,Y)

OFFSET	CONTENTS MSB (BIT 15)	LSB (BIT 0)
0	BIT 7 OF ALL DII(T) VALUES (L = 15 TO L = 0)	
2	BIT 6 OF ALL DII(T) VALUES (L = 15 TO L = 0)	
4	BIT 5	
6	BIT 4	
8	BIT 3	
A	BIT 2	
C	BIT 1	
E	BIT 0	
10	BIT 15 OF ALL DII(T) VALUES (L = 15 TO L = 0)	
12	BIT 14	
14	BIT 13	
16	BIT 12	
18	BIT 11	
1A	BIT 10	
1C	BIT 9	
1E	BIT 8	
20	BIT 23 OF ALL DII(T) VALUES (L = 15 TO L = 0)	
22	BIT 22	
24	BIT 21	
26	BIT 20	
28	BIT 19	
2A	BIT 18	
2C	BIT 17	
2E	BIT 16	
30	BIT 31 OF ALL DII(T) VALUES (L = 15 TO L = 0)	
32	BIT 30	
34	BIT 29	
36	BIT 28	
38	BIT 27	
3A	BIT 26	
3C	BIT 25	
3E	BIT 24	

SPECIFICATION

Vital processor V

The present invention relates to apparatus and techniques for performing vital processing using both vital and non-vital hardware. The invention finds particular utility in the railroad industry to replace vital equipment used for safety purposes, although the invention is widely applicable where ever digital processing must exhibit vital or fail-safe characteristics.

This application is directed to matter which is described in the specification of Patent Application No. 8428580 and is divided out of that application.

In order to provide for rapid and orderly vehicle movement while at all times respecting the overall safety requirement, the railroad industry has evolved a control and communication system. The control problem can be analyzed in terms of sensing real time conditions in a region of the right of way (present vehicle position, direction of motion, and condition of equipment, such as switches, signals, etc.) and based on a set of predetermined constrains imposed by the layout of the physical plant, determining what changes in equipment condition (e.g. switch position, signal condition, etc.) can be safely made to allow a vehicle to progress in its intended direction of motion. Once these decisions have been made, appropriate control signals are formulated and communicated to the actual physical plant to effect the desired changes.

Although safety is considered at every stage of information and communication processing, the rail-road industry's perception and practice has been that satisfying the safety requirement at every stage in the process is unnecessary and unduly complicates the equipment. Accordingly, in practice it is only the field equipment, which translates commands into physical manifestations (throw switch, clear signal), which is designed to meet vital or fail-safe characteristics. At earlier stages in the information and communication processing, while safety is always considered, failures in equipment employed in this earlier stage of processing need not exhibit fail-safe or vital qualities. Rather, the vital or fail-safe characteristic is imposed at the very end of the control chain, e.g. at the signals and switches themselves. This has allowed the railroad industry to modernize the majority of their plant by the use, for example, of solid state circuits and digital processing without necessarily requiring that this modernized equipment exhibit vital qualities.

Nevertheless, imposition of vital design results in a vast quantity of expensive, relatively slow, bulky equipment. There is naturally a desire to eliminate these deleterious characteristics.

At the same time, the decreases in cost for digital processing equipment (e.g. the ubiquitous computer on a chip) has generated a strong desire to employ this very capable, space economical, power economical, decision making component. For a host of reasons, it has been impractical to require that the design of these microprocessors follow the vital design techniques evolved in the railroad industry over the last 100 years. Accordingly, the industry has been searching for some technique (particularly software) which could be used to transform the admittedly non-vital microprocessor into a vital system.

Solution to this problem would result in numerous advantages to the railroad industry. It would simultaneously allow the application of cheap, fast, space saving, power saving and very capable devices for replacing the bulky, slow, electromechanical vital devices which had been employed in the past.

Although control of a railroad or a portion thereof requires the solution of many different control problems, all these different problems can be generalized into a single set of characteristics. The requirements are:

1. Sensing inputs in real time (the majority of the inputs are digital in nature, and to the extent that there are any which are not digital in nature, they can be transformed into digital inputs);

2. Deriving from these real time inputs a set of real time outputs for the control of different components in the railroad plant; where

3. The relation between these inputs and outputs is defined by one or more logic equations which can be rigorously defined in advance.

It would be inadequate for such a device to be merely capable of vitally solving the equations referred to in item 3, because the vital characteristic has got to cover not only the solution of logic equations, but sensing of the inputs and checking that the outputs presented to the railroad plant are in fact those outputs which have been derived by the solution of the logic equations.

Others in the field have attempted solutions to this problem, with different success; some of these solutions have applied traditional Ebp techniques. These solutions include:

A. Providing two identical digital processors each executing an identical program and providing that the processors execute their identical program simultaneously in time by providing for synchronization therebetween, and finally providing some means for comparing the results produced by each of these processors (and in some instances, internal intermediate results as well);

B. Providing two different digital processors solving the same problem in two different fashions (two different programs). In this case there is no need for synchronization since the

differences in processor and program characteristics necessarily result in differences in internal machine states; checking in this solution is only at the level of ultimate outputs.

An entirely different solution has been proposed for certain aspects of the problem related to communications. See, for example, Sibley U.S. Patent Application S.N. 273,299 filed June 15, 1981, entitled "Vital Communication System for Transmitting Multiple Messages". In this solution, it appears externally that there is only a single processor solving a single program; internally, however, in a time multiplexed fashion, the single program includes at least some diversity in that at least critical portions of the solution produce check words. The result of the single processor is provided in two forms, the first form is the outputs destined for the real world, and the second form is a series of check words which by their number and content perform a tell-tale function indicating the particular logic path followed by the program in the solution of the logic problem. Associated with the first processor (or vital processor) is a second processor (a vital driver); note that this is different from the solutions A and B noted above because the second processor is not at all concerned with the solution of any problem related to the real world environment. Rather, the purpose of the second processor is merely to review the number and content of the check words produced by the first processor. Only if the second processor indicates that the check words, by their number and content, verify the accurate execution by the first processor, will the real world outputs of the first processor be allowed to become effective. In order to close the loop, this solution has employed one or more techniques to verify that the input function has been performed vitally (that a closed contact, if present, is actually sensed, and that the representation within the first processor of this closed contact is indeed a representation of a closed contact) as well as checking that the potential outputs which the first processor indicates it will make effective if allowed, are in fact those outputs which flow from the solution of the logic equations effected by the first processor, e.g. is the output really dictated by the internal processes of the first processor, or does the output merely reflect a failed component?

Since the input information is essentially digital, as is the output, a very real difficulty is the need to verify that the single bit representation of this input which is being sensed or the output which is being checked, is appropriate; specifically that the input representation sensed by the machine, or the output representation being checked by the machine, has not been masked by a failure. Although all failure mechanisms have not been rigorously defined, two of the failure mechanisms which are well known are the "stuck bit" (where a bit is stuck in one of its two conditions) and the shorted terminal (where one terminal is shorted to another). Prior examples of techniques for overcoming these failure modes are illustrated in Sibley U.S. Patent 4,365,164.

Another difficulty which must be overcome is a byproduct of the presence within typical microprocessor systems of memory. The memory function presents at least two problems, data stored in the memory is going to be used in one or more intermediate processes, and even assuming that the data which had been stored in the memory was correct at some time in the past, how do we know that the data is still valid when it is being used? Furthermore, and also assuming that the data which is stored in the memory was and is correct, how do we know that the data we have extracted from memory is the data which we desire, and is not the result of some failure in an addressing mechanism?

One solution to the first problem is described in co-pending U.S. Patent Application S.N. 241,819. filed March 9, 1981 and assigned to the assignee of this application. This technique requires that once data has been used (or the last time it has been used) the data is destroyed. To ensure that data destruction has actually been carried out, each process which relies on the presence of current data includes an initialization routine solely for the purpose of checking that the data previously resident in the memory location, area or region, has in fact been destroyed. This initialization process produces one or more check words. The check words so produced are actually shipped over to the vital driver (the other, or checking processor) and unless the check words are correct (proving that old data had previously been destroyed and the results being checked are truly the result of current data) the checking processor will not produce the correct result which will not allow application of the vital processors' outputs. The whole system is arranged so that disallowance of outputs produces an entirely safe condition (albeit not necessarily the most efficient condition–all signals to stop). Furthermore, the check word using technique is arranged such that neither the vital processor nor the vital driver has stored therein the "right" answer. The presence of the "right" answer stored somewhere in machine memory raises the possibility that the "right" answer will be derived from memory and not necessarily reflect the appropriate checks. Therefore, in this and all other uses of check words for verification techniques, we must assure that the "right" answer is not available to the machine except by the intended processing.

Summary of the Invention

The invention provides a new solution for problems previously solved in the past, as well as providing solutions to those problems which have apparently been insoluble, all with a view

toward providing vital characteristics in an admittedly non-vital digital processor.

The present invention is particularly, but not exclusively, intended for application in an environment including five different elements. Thus, referring to Fig. 1D of the accompanying drawings (which is an overall block diagram showing one implementation of the invention) two of these elements are input and output devices 1 and 2. The input devices 1 are arranged to provide appropriate input signals for processing, the input signals corresponding to that information which is necessary in order to produce the desired output information. The output devices 2 have two functions, firstly they translate the signals representing output information as provided by a primary processor 3 into appropriate format to actually control the physical devices. The output devices 2 are arranged so that they are conditionally controllable in response to output information from the primary processor 3 in such a fashion that they do not actually control real world devices, but in their conditionally controlled condition, can be checked to provide additional input information to the primary processor 3-this additional input information consists of sensing the actual condition of the output devices. The information is used by the primary processor 3 to derive check words which, by their content, allow a comparison to be effected between the conditionally controlled condition of the output devices 2 and the information produced by the primary processor corresponding to the desired condition. A third element is a primary processor 3; this can be a conventional microprocessor which is provided with the software described hereinafter. The primary processor 3 has at least two different types of inputs, and two different types of outputs. One necessary input is provided by sensing the condition of the input devices 1. One type of output is information destined for conditional control of the output devices 2. The second form of input is determined by the conditionally controlled condition of the output devices 2. Finally, the second form of output is a time sequence of check words. Thus, software run by the primary processor, in addition to producing the information necessary to conditionally control the output devices 2, produces a sequence of check words which, by their number and content, perform a telltale function indicating the processing logic carried on by the primary processor 3.

The check words are destined for the fourth element, which is a vital relay driver 4 (or VRD). In implementation, the VRD 4 can be another microprocessor. Its sole function is to receive the stream of check words produced by the primary processor, and from that stream of check words, produce, if the check words by their number and content indicate faultless processing, a relatively unique signal which is not available in the apparatus from any other source. In an embodiment of the invention which has actually been constructed, this unique signal is a modulated square waveform of selected duty cycle, repetition and modulation rate. The unique signal is provided to a detector 5. The detector 5 merely responds to the repetition and modulation rate and duty cycle of the unique signal and produces a second relatively unique signal in the event that the repetition and modulation rate and duty cycle of the signal produced by the VRD 4 is (within some tolerance) what the detector 5 is designed to respond to. The second relatively unique signal, can for example be a particular DC voltage which is unavailable from the apparatus from any other source. This second relatively unique signal is provided to the output devices 2, and when so provided enables the conditional controlled output devices to actually control the real world physical devices.

A particular embodiment of the VRD 4 and detector 5 are described in co-pending application entitled "Modular Output Driver for Vital Processor Systems" (8428579 filed herewith and assigned to the assignee of this application, the disclosure of which is incorporated herein by reference.

Since vital operation depends on each component, the input and output devices as well as the detector 5 are constructed with vital techniques. An important characteristic of the invention is that the processor 3 (which must also exhibit vital characteristics) is made to exhibit those characteristics, not merely by its construction, but by the software which is run.

For the purpose of sensing inputs, and/or checking outputs, each input and/or output function is provided with a unique multi-bit name, i.e. a name which is different from the name used by any other output or input. The "name" is derived as follows. In order to sense the input and/or check the output (typically the condition of a contact–either closed or open) a sense circuit is provided for each such input and/or output. The sense circuit has two inputs and an output. One input to the sense circuit is the condition being sensed, e.g. the input or output. Another input is driven by a multi-bit signal. The sensing circuit is arranged such that if the condition being sensed is in one of its two states, the driving bit pattern at one input is reproduced at the output in its inverted sense, whereas if the condition being sensed is in its other state, then the output produces a null value. This arrangement satisfies the necessity for stuck bit prevention since the multi-bit driving pattern is a combination of 1's and 0's which will readily detect a "stuck bit". However, this potential solution itself presents two additional problems.

Since one of our requirements is nt to have the "right" answer stored anywhere in the machine, how do we generate the driving bit pattern to produce the unique name of the input or output, without having that unique bit pattern stored in the machine? The second problem relates

to the representation of the condition being sensed when in its other state, e.g. we do not want to maintain any vital information as a null string. These problems may be solved by methods and/or apparatus in accordance with the invention as follows.

The driving bit pattern for sensing inputs or verifying outputs is derived from a multi-conductor data bus. However, each different sense circuit is associated with a different conductor in the multi-conductor data bus. We drive the multi-conductor data bus with a sequence of multi-bit data units, each unit having a number of bits equal to the number of conductors in the bus. Accordingly, these data units are serially presented to the data bus. However, since each sense circuit is connected to only one conductor of the data bus, each different sense circuit sees only one bit (and the same bit) of each data unit. If the condition being monitored is in one state, the sense circuit produces an output which is the complement of its input, if the condition being sensed is in its other state, then the sense circuit procuces a null value. The outputs of the sense circuits are placed back on the data bus and applied to a series of shift registers wherein each shift register is dedicated to a particular conductor of the data bus (and hence associated with a particular sense circuit). After applying a number of data units to the sense circuits equal in number to the length of each shift register, the shift registers are read out in parallel, one after the other. This operation transposes the unit serial, bit parallel driving data to bit serial, unit parallel sensed data. The driving data units were presented from the machine memory in bit parallel, unit serial order. For each condition being sensed, the broadside output of the shift registers is now unit parallel, bit serial. The output of the shift registers is now stored in machine memory; as a result of this transposition the "name" of a particular condition or sensing circuit is not available to the machine. While the driving data bit pattern is stored in a table, since it is stored in a bit parallel, unit serial order, the name which is unit parallel, bit serial order, is not available.

The second problem is solved by actually assigning two names to each different input being sensed, or output being verified. Both names are unique, one is a "true" name and the other is a "false" name. The "false" name, since it will not be used except to establish a restrictive condition, need not be vital and thus can be stored and available to the processor. Each time the machine detects an input being sensed and/or output being verified which results in a null word, the "false" name for that function is substituted for the null value. Thus, we have solved both problems and now have a technique for sensing the binary condition of an input in a vital fashion resulting in a unique name for the "true" state of the binary condition as well as a unique name for the "false" state of the binary condition. As will be seen below, the uniqueness of these "names" can be used to verify that a data substitution has not been effected. Furthermore, because of the characteristics of each "name", it can be checked to determine that not only the sensing circuit, but all the other equipment in the chain producing the output of the sensing circuit is operating appropriately. To this end, each "name" is a code word in a set of code words. For example, in one embodiment of the invention, each "name" (true and false) has a bit length of 32 bits. This 32-bit length is broken up into three fields, the least significant bit indicates the binary condition, e.g. 1 or 0 (for true and false, respectively, this is the value bit or true/false bit). The next 13 bits are a unique 13-bit combination (the true combination is the complement of the false combination) and the last 18 bits are determined from the code rules to ensure that the entire word is one of a small set of 32-bit words complying with the code. This has a number of significant advantages. For example, conventional error detecting can be used to determine whether or not any 32-bit word is a valid code word, invalid code words may be the result of some error. If at any stage in the processing an invalid code word is detected, then the false designation for that word is substituted with two effects. Firstly, safety is not compromised (since the false word will result in a more restrictive condition and therefore satisfies vital requirements), and secondly, the substitution enables the subsequent processes to continue by ensuring that the results produced by processing produce valid code words themselves.

This describes a first "channel", for satisfying diversity we provide a second "channel" as follows. Each input being sensed, or output being checked has a second true/false name pair, also 32 bits long, but in this second channel, these names are part of a second, different, 32-bit code word set. Sensing of inputs and outputs is effected in both code sets, and some processing proceeds in parallel. This provides two levels or channels which can later be checked and if an error is determined at either level, the false or most restrictive word is substituted for the result.

Accordingly, we maintain a pair of buffers (one for each channel) in which we store words resulting from the sensing process. Before storing the words, we can effect a non-vital test on the sensed words to ensure that they are members of the appropriate code word set. Null words, or any other word which is not a member of the appropriate set, are converted to the corresponding false word for the associated function (function is another name for a particular input or output).

Because time is an important parameter in the control problem, we target the processor functioning to be effected in a unit time (in one embodiment of the invention this unit time was

one second). Knowing this parameter, then we can judge the effect on safety of delaying the change in an input; clearly some are so time sensitive that any further delay cannot be tolerated. However, there are input functions in which further delay can be tolerated, for instance, signal lamp repeater inputs. For those inputs which can be delayed without compromising safety (we will call this parameter a forgiveness parameter), we can increase the reliability (not the safety) of the entire system. Since sensing a condition takes a definite amount of time (measured in milliseconds), we can expect that some sensed words will be corrupted not because of an error but merely because the condition being sensed has changed at some point during the actual sensing process. If that input was a critical determinant of a particular output, then the rules we have thus far proposed (substituting a false value for a corrupted value) may require changing the output until the input is correctly sensed. However, for those input functions which we can determine a priorinot to have a significant time sensitive safety factor, we can build some forgiveness into the system by allowing it to use a previously, correctly sensed, value for some short period of time on the assumption that the condition causing the corruption in the sensing function will itself exist for a shorter period of time. To this end, we have identified an additional parameter related to each input function, called cycles of forgiveness, and thus we have in an embodiment of the invention actually constructed, functions with zero, one and two cycles of forgiveness (clearly use of three levels of forgiveness is not essential). To implement this, we triple the size of the buffer and partition it so that those functions with zero cycles of forgiveness directly load the buffer section (DIN) which will be used on the immediately succeeding cycle of processing. On the other hand, for functions which are defined as having one cycle of forgiveness, we may not load this section directly, but load a section (DIN B) one removed from that section which will immediately be used. Similarly, for functions with two cycles of forgiveness, we may not directly load either DIN or DIN B, but rather a third section (DIN A). We adopt the following rules. Whenever a function is accurately sensed, it is loaded into all three sections of the buffer, e.g. it is used immediately, and the other two buffer sections provide some "memory" for this function. If a function is sensed incorrectly and has at least one cycle of forgiveness, then the DIN buffer is not loaded with the false value produced as a result of the incorrect sensing. Rather, the DIN B or DIN A buffer is loaded (depending on whether the function has one or two cycles of forgiveness). At the end of any cycle, the contents of the DIN buffer is cleared, a check word is generated to prove that the clearing has been accomplished, the contents of the DIN B buffer are shifted down one section (to the DIN buffer), the DIN B buffer is cleared, another check word is generated to prove that this clearing has been accomplished, the contents of the DIN A buffer are shifted down to the DIN B buffer and the DIN A buffer is cleared, and finally another check word is generated to prove that the DIN A buffer has been cleared.

To ensure that a word intended to a specific buffer section (DIN A, DIN B, DIN) is not erroneously written or read from the wrong buffer, a constant unique to the buffer section is added to the word on writing such that a word in DIN A has the value WORD, whereas in DIN B, the identical word has the value WORD+A and in DIN the same word has the value WORD+A+B. Thus, if a word is erroneously read from DIN A or DIN B rather than DIN (or read from DIN A rather than DIN) then it will not have the expected value and as a consequence an error will be detected in subsequent processing (as will become apparent hereinafter).

Of course, the purpose for assembling all this information is to solve the control problem. The control problem is embodied in a set of logic equations, each relating a different output or function to one or more inputs and/or inputs and outputs. Each equation is in the form of a sum of one or more terms, each term itself is made up of the product of the value of different functions. The equations are evaluated in a specified order in the following fashion. An equation is selected for evaluation, and the first term is evaluated, this evaluation can either be true or false (evaluation of a term made up of a product of one or more functions is merely the logical product of the value bits of the different functions in the product); if false, the next term is evaluated and so on until a term is found which is true, making the function defined by the equation also true.

An example will illustrate. Assume we wish to evaluate the following equation defining the condition of an output N:

$OF_N = IF_1.IF_3 + IF_7.IF_2 + IF_1.IF_2.\overline{IF_4}$.

Thus, the state of the output N (OF_N) is defined by three terms, term one is the product of inputs 1 (IF_1) and 3, term two is the product of inputs 2 and 7, and term three is the product of inputs 1, 2 and the inverse of input 4. If any evaluation of any term is a logic 1, then the output N is also a logic 1.

If no term is found making the function defined by the equation true, then the false word associated with that function is stored in the appropriate slot in an output buffer. This initial evaluation proceeds in a non-vital fashion by scanning the true-false bit of the value of each function in a term. In this fashion, the first term in an equation which is true is readily determined. Of course, if no term in the equation makes the function true, then the false value is

assigned to the function and the false value is stored in the appropriate place in an output buffer.

Once the first term which makes the function defined by the equation, true, has been identified, then processing switches from non-vital to vital in the following fashion. Our assignment of "names" includes the output functions as well as the input functions. Our vital processing is arranged so that we actually "compute" the value of the output function from the present value of those functions in the term we have selected. This "computing" has unique or novel characteristics. We must be able to arrange the computation so that computation of the output function value in accordance with any term will produce the identical and predetermined result. As is described below, we rely on the finite state characteristic of a maximal length feedback shift register for this characteristic. The value of each function in the term which has thus been selected is "added" in a polynomial divider with the value of each other function in the term. The value of each function is determined by the value for the corresponding function in the DIN buffer. The result of this processing is the value assigned to the output function. By design, the value of the output function is a 32-bit code word in the first code set. The value of the output function can be checked non-vitally for validity at this point by ensuring that its value is truly one of the limited 32-bit code words in this first code set. After checking, it can be placed in an output buffer for later use. Since the output function must have an expected value which is identical regardless of the term in the equation which makes it true, the polynomial divider, employed in the "adding" process, is precharged with a unique quantity for each different term in the equation to ensure that regardless of the term employed, the output function will have the identical value.

Since this process or computing the value of an expression is an important characteristic of the invention, the requirements should be explained.

Since we have already assigned a "name" pair to each function, including the output functions whose value is to be computed, the result of the computation must be identical to the expected result; this characteristic can be relied upon in order to prove faultless operation. First, the result may be a member of the code set, and a test is made after the computation for this feature. While this is a non-vital test, the probability of incorrectly arriving at a result which is a member of the right code set is about 4×10^{-6} (1 part in 2.6×10^5).

The vital test comes at a later stage when the computed result becomes a component in production of a check word using a similar technique. The checkword is tested for correctness by the VRD and failure of the test will prevent application of the resulting output to any real world device.

The actual computation is effected in a polynomial divider which includes a shift register with a controllable feedback network, particularly a feedback network making the feedback shift register capable of producing a maximum length sequence. Selecting the condition of the feedback network personalizes this divider to a particular code set. The computation is divided into an intialization function (which is performed once per computation) and a two-step sequence which is performed for each factor in the product being computed.

Once the shift register feedback network is personalized, an initializing constant is selected and loaded broadside into the shift register. The initializing constant has a bit length equal to the length of the shift register (although it is not essential), and there is a separate initializing constant for each term of each equation. Thus, necessarily selecting the correct initializing constant can only be effected once the term being computed has been identified. Once the initializing constant has been loaded into the shift register, the shift register is stepped a number of times I (1 bit shift per step). Because of the feedback network, the result is not a simple shift of the contents, but depends on both the bit pattern of the constant and the condition of the feedback network. The shifting concludes the initialization function.

The two step sequence effects the following functions:

a) The factor to be 'added" is added, modulo 2, to the shift register. This is an independent operation at each bit position, the bit in any shift register position is added, modulo 2 to the corresponding bit of the factor. Addition modulo 2 obeys these rules, $1+0=1$; $0+1=1$; $1+1=0$; and

b) This shift register is again stepped a number of times A (again one bit shift per step).

After the two step sequence is performed once for each factor, the result is the bit pattern in the shift register.

In an embodiment which has been constructed, the number of shifts I and A, are equal to each other and equal to nine. Both selections ($I=A$, and $I=9$, $A=9$) are arbitrary, so wide variations are possible. While we may eliminate shifting after the load (i.e. I may be equal to zero), however, the shifts after the "add" are desirable (i.e. A should be at least one). As implied herein, the embodiment contructed used a hardware shift register and feedback network. It is certainly conceivable that an equivalent operation could be carried out in software, eliminating the hardware requirements.

We also have values in the second channel buffer (values in a second code set) for each

function in each term of the equation. Having selected the term in the equation which made the output true (for the first code set or channel), we employ the identical term, but using the values from the second code set (the second channel) and again evaluate the output function. The result, if valid, will be a code word for the output function in the second code set. We can then verify non-vitally that the function value is a valid code word in the second code set. If the output function does not have a value in both channels satisfying the code rules of the respective channels, then the output function is considered false and thus the false value for the function is assigned and loaded in the appropriate locations in the output buffer.

By now, the protection provided by this type of evaluation should be apparent. By prearrangement, evaluation of any function (a logical combination of input values) should produce a value unique to that function. The correct result (which is checked for by the VRD) can only occur (except at a vanishingly small probability) as a result of flawless processing. If an input word is garbled, it will probably not pass the non-vital test and be replaced by the false (or restrictive) name–a safe failure. Even if the garbled sensed word is a valid code word (improbable, but possible), or if a correctly sensed word is mis-read or mis-written, the failure will not (except in a miniscule number of cases) produce the identical word (bit for bit for 32 bits) which is required. Unless every one of the 32 bits is the expected value (in both channel 1 and channel 2), the function evaluation will not produce the expected result. While this may not affect processing in the primary processor 3, it will be caught at the VRD4 and prevent application of inaccurately computed results. The error could be detected if the computed result fails the (non-vital) test effected prior to writing the evaluation results to the result buffer.

While examples thus far discussed describe how output function values are determined from input function values, it is worth noting that output function values may also depend on other output function values. However, of course, an output function value cannot be computed until all factors defining that particular output function have themselves been either sensed or evaluated.

Accordingly, the preceding operations have vitally sensed inputs (both by using a multi-bit driving bit pattern to produce a unique name for each function where the correct name for the function is not stored anywhere in the machine, although the machine is capable of checking to ensure that the value of the function is one of a limited set of values); this sensing is performed in two different channels using two different code sets. The sensed value, after it is checked, is loaded in an appropriate buffer slot. The buffer is divided into two halves, one for each sensing channel, and each half is further subdivided into three sections. If the sensed value is not a member of the appropriate code set, the corresponding false value may be loaded in either one of the three sections, depending on a "cycles of forgiveness" parameter which is associated with each different function. For those functions which have non-zero cycles of forgiveness, buffer management provides for some "memory" of a previously sensed correct value. Provision is made for guarding against misreading the buffer section in the following fashion. After each cycle of processing is completed, the buffer section which is actually used in solution of the control problem is cleared (this clearing is effected vitally and a check word is generated to ensure that clearing has actually been accomplished). The immediately adjacent buffer section contents are then transferred to the just cleared buffer section, but a constant is added to each value in the transfer process. The buffer section just operated on is then also cleared in a vital fashion (again generating a check word to prove that it has been cleared). Thereafter, the contents of a next buffer section are again transferred to the just cleared buffer section, again adding a different constant, and finally the last processed buffer section is agan vitally cleared. The use of the different constants ensures that misreading a buffer section will not go undetected for the value read will not be the value expected and the subsequent processing is arranged to ensure that an unexpected value will positively be detected.

The control problem is actually solved once the inputs are appropriately arranged through the use of a set of equations defining a value for each output function in terms of the sensed values and/or previously computed output functions. Each such equation is made up of a sum of terms, where each term in turn is made up of a product of different function values. The processing proceeds in stages. First, an equation is selected for evaluation, second, the first term in an equation is evaluated. This evaluation proceeds quickly in a non-vital fashion based on the true/false bit of each value. In this fashion, evaluation quickly selects the first term in an equation which makes the function defined by that equation true, or quickly determines that there is no term in the equation which makes the function true. In the latter case, the false value for the function is assigned to the function and that false value is stored in an appropriate section in an output buffer. If, on the other hand, the value of the function has non-vitally been determined to be true, then processing switches from a non-vital to a vital mode.

In the vital mode, each value in the term making the parameter true is "added" in a polynomial divider to each other value in the term. The result of this "addition" in the polynomial divider is a value for the output function. Since by design the value for the output function should be identical regardless of which term in the equation makes it true, the polynomial divider

is precharged with a unique constant for each different term to ensure that processing will result in a single true value for the output function regardless of the term in the equation which makes the output function true. Furthermore, the values at the output functions are selected to be members of a small set of code words in the first code set. Based on this characteristic, we can check the value of the output function to ensure that it is indeed one of the small set of allowed code words. If it is not, then the false value for the function is assigned. If the output value for the function passes this check, then it is conditionally determined to be the correct value for the output function and is assigned to the appropriate slot in an output buffer.

Processing is not yet complete, since we have a second channel of data (in the first channel, the processing for which has just been discussed, each of the values is a code word in a first code), in the second channel all of the values are code words in a second, different, code set.

Processing in this second channel proceeds as follows. We do not search the terms in the equation for one making the output function true in this second channel. Rather, we use the identical term as the one employed in the first channel processing. We extract the values for the functions in this particular term from the second half of the buffer (corresponding to the second channel) and these values are "added" in our polynomial divider. Prior to effecting the "addition" the polynomial divider had been previously "pre-charged" with a constant related to this particular term in the equation. The result should be a second unique value for the function which should be a code word in the second code set. After "addition" in the polynomial divider, the resulting value is checked to ensure that it is indeed a code word in the second code set. If it is, then this value is stored in an appropriate slot in an output buffer. On the other hand, if it turns out that the value so computed does not belong to this second code set, then an error has been detected. The false value is assigned to the parameter both for this second channel processing, as well as correcting the first channel processing by assigning the false value to the function for the first channel as well.

In a similar fashion, values for each of the functions defined by each of the other equations is similarly effected. At the conclusion of this processing, we have in the output buffer a succession of values for each output function.

It should be emphasized that while we require each function evaluation to produce a unique value, we have no way, at this point, of verifying that the unique value has really been computed. All we can determine at this point is that the parallel function evaluation (one in each channel) has produced values consistent with the respective code rules for the channel. Later processing will be used to verify that the values so computed are the unique expected values.

The output values are now transferred, at least conditionally as follows. The true/false bit of each channel 1 value is sampled non-vitally and the corresponding output bit is set accordingly. Because this is a non-vital transfer, the process must be checked to ensure that even at this late stage in the processing, we have not converted a false to a true or a true to a false. The checking proceeds as follows. The channel 1 and channel 2 expressions for each function are combined and the result of this combination is stored. This result will be employed in subsequent processing. What we want to assure is that we can detect an open contact (or an off output) for each output bit unless the combined channel 1 and channel 2 data allows the output bit to be true. Before describing this checking process, a brief word is in order about timing.

The time required in available equipment to perform this type of processing for a reasonable number of input and output values will take at least 0.5 seconds. We estimate the processing time for the solution of the control problem at a medium sized interlocking which has about 100 vital inputs, 125 vital outputs and 350 equations. However, within this even short period of time, the state of a relay can change (typical relay pick times are about 150 ms). Accordingly, if we are operating in a vital environment, we must ensure that a relay does not change state without being detected. Obviously, we cannot incorporate this protection within our minimal 0.5 second cycle time. To solve this problem, we divide the processing up into a main cycle, nominally 1 second in duration, and a plurality of rechecks (each less than 50 ms in duration and occurring every 50 ms) and thus there are 20 rechecks during a main cycle. It is during the recheck processing that the output function and checking takes place and actually the main cycle processing is fit in between the recheck process. In this fashion, the recheck processing is used every 50 ms to prove that those outputs in their permissive state are supported by the corresponding expressions in channel 1 and channel 2 being both evaluated as 'true'.

The main component in this recheck processing is in many respects similar to the processing described at input sensing. Each output bit is connected to a sense circuit (called an absence of current detector, or AOCD) and is driven by a 32-bit value. The output of the AOCD is the complement of the input if the output is off or false. Thus, null values are returned for output bits which are on. However, these bits must have expression results which allow them to be true.

Thus, after the recheck is completed we have evidence (in the form of check words) that can be used by the VRD to verify that for every output function conditionally in the 'true' state, our function values in both channels allow the true state, and that every other function is in the

'false' or non-permissive state.

Thus, the tentative values for output function which have been computed are used along with the conditional condition of the output circuit to develop check words. These check words, if correct, indicate that both the computed value of the output as well as the condition of that output are both unique and in correspondence. Any other condition will be detected by the VRD 4 and not allow application of the output signals to the real world.

According to the invention a method of computing a multi-bit binary value of significance from two or more input multi-bit binary values comprises:

a) providing a feedback shift register, with plural stages, arranged to provide maximal length sequences

b) preconditioning said feedback shift register by controlling each stage thereof to attain a condition identical to a corresponding bit of a predetermined bit pattern,

c) shifting first one of said input multi-bit binary values into said feedback shift register, and

d) repeating said step (c) for each other of said input multi-bit binary values.

Said step (c) may involve exclusive OR'ing said first multi-bit binary value, bit by bit, with a corresponding stage of said shift register.

Step (c) may include the step of shifting the feedback shift register a given number of times after said one of the input multi-bit binary values is exclusively exclusively OR'ed into the feedback shift register.

Step (b) may include the step of shifting the feedback shift register a first given number of times after each of the stages is controlled and step (c) include the step of shifting the feedback shift register a second given number of times after said one of the input multi-bit binary values is exclusively OR'ed into the feedback shift register. These first and second given numbers may be equal.

Each of the input multi-bit binary values and the predetermined bit pattern may have a bit length equal to the number of stages of the feedback shift register.

The method may include the further step of testing the resulting bit pattern to confirm it includes two fields, a first information field, and a second check field related to the first information field by a predetermined code rule. Each of the input binary values may also include a first information field and a second check field, which second check field is related to the corresponding first information field by the identical code rule.

Brief Description of the Drawings.

The invention will now be further described by way of example only in such detail as to enable those skilled in the art to make and use the same in the following portions of the specification when taken in conjunction with the attached drawings in which like reference characters identify identical apparatus and in which:

Figures 1A, 1B and 1C are functional block diagrams useful in explaining the input sensing functions;

Figure 1D (already referred to) is an overall block diagram of the various components of the invention in a typical implementation;

Figures 1E–1G are block diagram of the polynomial divider in generic form, and as personalized for the two different channels of processing;

Figures 1H and 1J are useful in describing the AOCD and its method of operation;

Figure 2 is a functional block diagram useful in explaining the manner in which the values produced from the input sensing function are checked and stored for later use;

Figure 3 is a functional block diagram useful in explaining the evaluation function;

Figures 4 and 5 are block diagrams of the input equipment;

Figures 6 and 7 represent functional block diagrams of the output and output check process;

Figure 8 illustrates how the polynomial divider 509 of Fig. 3 is connected as peripheral to the primary processor 3 of Fig. 10, and specifically breaks out the control bits (also derived from the primary processor 3) which are used to operate the polynomial divider 509 in its various modes;

Figure 9 is a schematic of the polynomial divider, specifically the shift register and its peripheral apparatus arranged to operate in the evaluation mode;

Figure 10 is a similar figure illustrating the shift register and peripheral apparatus arranged to operate in the check mode;

Figures 11–14 illustrate the various tables of application data, including DIHEAD-Fig. 11; DIGRXY-Fig. 12, DIS1XY-Fig. 13 and TIL1 (L,X,Y)-Fig. 14.

Detailed Description of Preferred Embodiments.

Simplified Description.

Fig. 1D is an overall block diagram illustrating typical application of the invention. As shown in Fig. 1D, the environment comprises five elements, e.g. input devices 1, output devices 2, a primary processor 3, a VRD 4 (vital relay driver) and a detector 5. The present application is

particularly concerned with elements 1–3; one implementation of elements 4 and 5 is disclosed in detail in our co-pending application 8428579 S.N. filed herewith. As is shown in Fig. 1D, the input devices 1 respond to physical inputs (this can be from such devices as tracks switch condition (normal, reverse, etc.), track occupancy condition (occupied or unoccupied), traffic direction indication, etc. The function of the input devices 1 is to translate the condition of these physical devices into signals capable of being sensed by the primary processor 3. The primary processor 3, which for example can be a conventional microprocessor, includes as part of its function the sensing of the condition of the input devices 1. The primary processor 3 uses this information, in a manner to be explained, and produces output information which is coupled to output devices 2. A second class of input information for the primary processor 3, is provided by checking the condition of the output devices 2. In addition to its function of sensing real world information, generating output information for the output devices 2, and checking the condition of the output devices 2, the primary processor 3 also develops a second type of output information. This second type of output information is a time sequence of check words. The check words, by their number and content, perform a telltale function indicating the logic path followed by the primary processor 3 through its software, and includes a telltale indicating whether or not the output devices 2 are in a condition which corresponds to the condition to which they should have been controlled if they had been following the output information provided to them by the primary processor 3. This stream of check words is not at all evaluated by the primary processor 3, rather it is provided to the vital relay driver 4 (or VRD), which can also be implemented as a conventional microprocessor. The sole purpose of the VRD 4 is to evaluate the sequence of check words, and produce a single output. The single output can take a variety of forms, its function is to be an almost unique signal, at least one that is not available for many other apparatus. In an embodiment of the invention which has been constructed, this relatively unique signal takes the form of a modulated square wave of selected carrier frequency, repetition rate and duty cycle. The carrier frequency, repetition rate and duty cycle of the output of the VRD 4 depends on the number and content of the check words, as well as their rate of production. The VRD 4 is arranged so that only if the check words form a sequence whose rate, number and content indicates flawless processing by the primary processor 3, will the output take the form of the modulated square wave with the desired carrier frequency, repetition rate and duty cycle.

In the event that the appropriate output signal is produced, the detector 5 (which is designed to respond to this particular waveform of the appropriate parameters within a given tolerance) will produce a second relatively unique signal. The second relatively unique signal can, for example, be a particular DC voltage. The second relatively unique signal is also relatively unique because it is not available from any other apparatus. This particular level of direct current is necessary to enable the output devices 2 to actually control the physical outputs to which they are connected. Thus, while the invention includes a pair of processors, they are operating on distinctly different information and producing distinctly different results, in fact the VRD 4, and the associated software, has no relation to any real world condition. The VRD 4 merely sees, in essence, a sequence of numbers, and processes those numbers to produce a corresponding output. The VRD 4 has no way of even "knowing" the effect of the output it produces.

The invention can be applied in a variety of circumstances. It can be applied in a quite straight forward manner to control a selected region of the railroad right-of-way, e.g. adjacent switches and signals (in which event the input devices 1 may include control information provided from an operator's keyboard, either directly or indirectly via some communication link), it could be one element (transmitter-or receiver) in a communication system. If the apparatus of Fig. 1D were acting as a transmitter, then the output devices would include a communication link, if the apparatus of Fig. 1D were acting as a receiver, then the input devices would include a communication link. The overall goal of the invention is to provide hardware and software so as to reduce the probability of an unsafe failure to a vanishingly small number notwithstanding the fact the intelligent element (the primary processor 3) is in fact a conventional garden-variety microprocessor or the like.

Error Detecting Codes and Use of Polynomial Divider or Feedback Shift Register

At many points in the processing, it is necessary to determine whether a particular word is a valid word, e.g. within a small group of words satisfying particular code rules. The manner in which these words are selected and detected, will now be explained.

Error detecting code are well known to those skilled in the art, they achieve their error detecting capabilities by adding onto a k bit message, r check bits, so as to form a word with n bits, wherein $n=k+r$. A set of linear algebraic equations can be used to calculate this set of r bits from the set of k message bits. While any set of equations can be used as the basis of a code, a significant subset is the linear block code, where an n bit word is formed from k message bits and r check bits. For ease of implementation, it is preferred to use a subset of linear block codes which are similar to cyclic codes. Cyclic codes are linear block codes which

satisfy an additional constraint. That is, cyclically shifting a cyclic code word by one place produces another cyclic code word. By extension, all shifts of a cyclic code word result in other cyclic code words. While in general linear block codes can be generated by using matrix multiplication, the cyclic codes allow the use of shift registers to produce and/or check for the presence of valid code words. Those skilled in the art will appreciate that the use of shift registers is simpler and faster than the use of matrix multiplication.

There is, in addition, a subset of cyclic codes which are systematic. In the systematic cyclic codes, the message bits always appear in predetermined positions, so that the check bits necessarily appear in other, predetermined positions.

A systematic cyclic code can be generated by the following steps:

1. Multiply m(X) [the message or information] by x^{n-k} (where n is the bit length of a code word, and k is the length of the message bit portion, so that $n-k=r$, where r is the number of check bits in the code word). This first step puts the high order bit in position $n-1$.

2. Divide the product $m(x).x^{n-k}$ by g(x) [where g(x) is the generator polynomial for the cyclic code] and keep the remainder; the division produces a quotient q(x) and a remainder c(x). The remainder must be of a degree less than the degree of g(x) so it can have at most degree $r-1=n-k-1$.

3. Add the remainder c(x) to the product $m(x).x^{n-k}$ formed in step 1 to form a code word (in this adding step, we are using binary addition equivalent to subtraction or exclusive OR'ing).

The code word constructed this way consists of two independent fields, a first field containing message or information bits only, and a second field (exclusive of the first field) which contains only check bits. The quotient produced in step 2 is irrelevant and need not be retained.

This coding processing can be implemented by using a feedback shift register which is connected to effect polynomial division. An arrangement that will simultaneously multiply m(x) [the message] to be multiplied by x^{n-k} [step 1] and divide by g(x) [step 2] is shown in Fig. 1E. As shown in Fig. 1E, a shift register of $n-k$ stages (c_0 to c_{n-k-1}) has a plurality of exclusive OR gates R, a different one at the input to each stage, and a GATE for input control. In addition, a two position switch (with positions A and B) is connected to an output line OUT. With GATE "on", and the output switch in position A, the k information digits m(x) are shifted into the register and simultaneously to OUT. As soon as the last message digit has been shifted in, the $n-k$ digits in the register are the parity check digits. By turning the gate GATE "off" and putting the switch in position B, the check digits can be shifted to OUT. The shift register shown in Fig. 1E is personalized for any particular generpator polynomial by selecting the condition (open or closed) for each of g_1 through g_{n-k-1} to match the corresponding generator polynomial coefficients.

In accordance with one aspect of the present invention the feedback shift register is arranged to provide maximal length sequences. Thus, code words are selected in such a way that the minimum distance (d) is maximized (the minimum distance is the distance between two valid code words, e.g. the number of bit changes that must be made to turn one code word into another valid code word). The minimum distance is important because it determines the number of undetectable errors. Any number of changes which are less than the minimum distance will be detected. In order to use cyclic codes, it is necessary that the degree of the generator polymonial be $r=n-k$. Furthermore, g(x) must divide x^n+1 so that the cyclic properties emerge. The effort to locate good cyclic codes is a continuing one and the resulting studies are tabulated in existing textbooks.

Input Function

Figs. 1A and 1B are useful in explaining the input sensing function. These figures schematically illustrate the apparatus and processing carried out in order to implement input sensing. As is shown in Fig. 1A, a plurality of input terminals 1–1 through 1–n are each coupled to one input terminal of a dedicated sense circuit 35–1–1 through 35–1–n, one sense circuit for each input circuit. An n conductor bus 30 is coupled to the drive input terminals for each of the sense circuits 35–1 through 35–N, a different conductor in the bus coupled to each different drive terminal. Each of the sense circuits also has an output labelled "SENSE", which is connected to an input terminal of a signature element 35–1. The particular pairing of inputs and outputs for the signature circuit 35–1 pairs outputs and inputs such that, for example, inputs 1 to N–1 are respectively connected to outputs 2 to N, with input N connected to output 1. Obviously, many other variations could be envisaged. The signature element 35–1 is useful in those cases wherein there are more than N input terminals, allowing groups of N input terminals to be uniquely identified by using different signature elements, e.g. 35–1, 35–2, etc. for different groups of input terminals. It should be apparent that the signature element 35–1 would have identically the same function whether it is coupled between the bus 30 and the drive terminals of the sense circuits or the SENSE terminals and the conductor 30.

In order to provide driving signals for the sense circuits, a source of a bit pattern, such as ROM 10 is coupled to the conductors of the bus 30, and the bit patterns stored in the ROM 10

can be selectively applied to the conductors of the bus 30 via conventional addressing arrangement 15. Preferably, the ROM has a bit width which is equal to the bit width of the bus 30. Bit patterns are applied to the bus 30 in what could be termined word serial order; however, since "word" may carry the connotation of 8 bits, and the present invention is not limited to using words of 8 bit, we will hereinafter refer to the bit pattern stored in the ROM 10 as existing in unit serial order wherein we have substitued the generic word unit for the specific term "word". Accordingly, as the ROM 10 is sequentially addressed by the addressing circuit 15, the bit patterns stored in the ROM 10 are applied to the conductor 30 in unit serial, bit parallel order.

Also coupled to the conductors of the bus 30 is a transposition arrangement including N shift registers 40, each shift register connecting to a different one of the conductors in the bus 30. The shift registers are loaded, bit serially, and are read out broadside. As is shown, the broadside reading of the shift registers is used to load a further memory storage arrangement, e.g. RAM 20.

Each of the sense circuits 35−1−1, etc. senses the condition of its associated input terminal in the following fashion. The input terminal can be in one of two conditions, either it carries a DC potential above a given threshold, or it does not. If the input terminal is in the first condition, then the SENSE output terminal reproduces in its inverted sense the pattern provided at the drive terminal, e.g. for example a 010 pattern at the drive input terminal under those conditions would produce a 101 pattern at the SENSE terminal. On the other hand, if the input terminal is in the other condition, then the SENSE terminal will produce a string of null values (e.g. logic 0) regardless of the input pattern applied at the drive terminal. Although the particular sense circuit is not essential to the invention, one implementation of the sense circuit is shown in Fig. 1C. The implementation of the sense circuit, however, must be 'vital', i.e. must have no failure modes which would allow an output simulating an on input without a DC potential at the input terminal.

In view of the foregoing, the operation can now be explained. At those times set aside for sensing input condition, the ROM 10 is addressed to provide a sequence of bit patterns on the bus 30 (in unit serial, bit parallel order) corresponding to the bit pattern stored in ROM 10. Each sense circuit is responsive to the bit pattern on a single one of the conductors in the bus 30. The sense circuit also directs output to a single one of the conductors in the bus 30. (In the absence of the signature circuit 35–1, these two conductors are identical, but in the presence of the signature element 35–1, they may be different.) After the first cycle of driving the bus 30, developing the sense pattern and shifting the pattern into the transposition element, each one of the shift registers 1–N has one bit stored therein derived from a different one of the sense circuits 35–1–1 through 35–1–N. After a second cycle, each of the shift registers has two bits stored, and after a number of cycles equal to the lenth of the shift registers, each of the shift registers are full. The shift registers are now read out broadside, one after the other, and the contents of the shift registers are transferred to the RAM 20 in the same order as the shift registers are read.

Fig. 1B shows more clearly the result of the operation of the apparatus shown in Fig. 1A. More particularly, Fig. 1B schematically identifies the equipment, e.g. ROM 10, bus 30, sense circuits, signature element 35–1 and the transpose arrangement 40 as well as the RAM 20. Fig. 1B, however, shows in some detail the bit patterns employed. As indicated in Fig. 1B, in the first cycle of operation unit 0 is read from ROM 10 in bit parallel order. Thereafter, unit 1 is read, and following unit 1, unit 2, etc. After the eighth unit, e.g. unit 7, has been read, then the shift registers and the transpose element 40 would be full if they were 8 bits deep. Thereafter, assume that the transpose element 40 is read broadside, one shift register after the other, to load RAM 20. Then the first shift register will produce the inverted bit pattern 00011111 which corresponds to bit 1 of the first eight units (assuming that the input terminal coupled to sense circuit 35–1–1 was in a logic 1 condition to allow the output to repeat the input, and of course also assume that the signature element 35–1 coupled its first input and output). The RAM 20, illustrating for bit b_2 an all 0 pattern indicates that the sense circuit 35–1–2 had an input terminal not in its logic 1 condition (also again assuming that the input and output position 2 of the signature element 35–1 were interconnected).

Although we have shown a square matrix, e.g. an 8 conductor bus 30 driving sense circuits with 8 data units, that of course is not essential to the invention. There is a relation between the number of conductors in the bus 30 and the number of sense circuits (there must be at least as many conductors in the bus as there are sense circuits or input terminals). However, the number of units which are used is related to the length of the shift registers and the depth of the RAM 20. For example, if the RAM 20 was 16 bits deep, and that was the length of the shift registers, then rather than using 8 units (units 0–units 7), we could have used 16, e.g. unit 0–unit 15, to develop 16–bit units in the RAM 20 rather than the 8 bits illustrated. Furthermore, the operation can be concatenated as follows. Assume that the shift registers are 16 bits long, meaning that after applicaton of 16 units by ROM 10, the shift registers are full. The registers can then be read broadside. This provides for 16 bits per input. However, this operation can be

repeated. We can arrange the addressing of RAM 20 so that the first and second 16 bits read from a single shift register are associated as a single 32-bit data unit.

Checking of the Input Sensing Function

In a preferred embodiment of the invention, the sensing bit pattern actually employed with any sense circuit is made up of two 16-bit units so that a single sensing word is 32 bits long. To allow ready checking of the sense word, each sense word is a code word in a limited code set. As described above, the sense word is broken up into three fields, the least significant bit is the T/F bit, the next 13 bits uniquely identify the input port, position or function. Actually, there are two different 13-bit fields associated with each input port, a true field, and the complement of the true field for those cases where the input port is in its off condition. These first two fields (T/F bit and 13–bit "name") can also be considered a first information field, since the two names are complementary through the first 14 bits. The following 18 bits comprise a check field. There are in fact two different sensing channels, using the same apparatus, but with different driving bit patterns. One results in sense words in a first code set CH 1 and a second channel results in sense words in a second code set CH 2. So in practice, we have two "name" pairs for each input (and output, as described below), one pair in each of two code sets or channels.

After the raw sensed data is stored in the RAM 20, a number of functions have to be performed prior to equation evaluation. Firstly, the words have to be checked to determine if they are valid words, during the course of this process, null words are replaced by the false value for the function. Secondly, as has also been described above, each input buffer is divided into three sections based on a parameter called cycles of forgiveness. In those cases where a corrupted word has been sensed, the false value will be stored, however, depending on the number of cycles of forgiveness, not all sections of the buffer will be written with a false value. The apparatus to perform these functions is functionally illustrated in Fig. 2.

Fig. 2 illustrates two sections of the RAM 20, TEMPI and TEMPI', corresponding to the first and second sensing channels. Furthermore, a storage area 25 is provided for storage of false values and a forgiveness store 26 is provided producing signals 0, I and II, corresponding to 0, 1 and 2 cycles of forgiveness. The raw data is applied to an all zero test 101 producing a YES or NO output, the first indicating that all bits of the word are 0's or null values, and the second output indicating that not all bits of the word are null. A polynomial divider test 100 is provided to check whether or not the word is or is not a member of the appropriate code set. The poly divider test 100 produces PASS and FAIL outputs, the first indicating that the word is indeed a member of the appropriate code set, whereas the second indicating that it is not. A plurality of logic gates 152 are provided to couple either the sensed raw data or the appropriate false value (from the false store 25) to a data bus 150. In the event that the all zero test 101 indicates that all bits are not zero, and the polynomial divider 100 indicates the word is in the appropriate code set, then the raw data is used; on the other hand, if all bits in the unit are zeros or if the polynomial divider test 100 indicates that the unit is not in the appropriate code set, then the false value, from the false store 25 is passed to the bus 150.

As indicated above, the input buffer is first divided into two halves, one corresponding to each input sensing channel, and each half is divided into three sections corresponding to, respectively, 0, 1 and 2 cycles of forgiveness. More particularly, the three sections in the first half of the buffer are DIN, DINB and DINA while the second half includes DIN', DINB' and DINA'.

In the event that a particular sensed word passes both tests, i.e. the polynomial divider 100 produces a PASS output and the all zero output produces a NO, then the sensed word will be used unchanged. However, in order to distinguish between the three sections in each half of the buffer, there is a constant (+A) difference between the representation of a sensed word in the highest section (DIN A), and the same representation in the immediately adjacent section (DIN B). Furthermore, there is a different constant (+B) between a representation in that section (DIN B) and the immediately adjacent lower section (DIN). To implement these rules, in the event that the tests indicate PASS•NO, then the sensed word is output from the bus 150 through a gate 161, and it is stored in buffer section DIN A. Thereafter, the same enabling signals couples that quantity through XFER 170 (wherein the constant +A is added thereto and it is loaded into DIN B). Thereafter, the same control signals allows the same quantity to be coupled through XFER 172 (where the constant +B is added thereto and it is stored in DIN). Accordingly, in the event of the PASS•NO condition, a quantity (Q_0–for example) is stored in DIN A; the quantity Q_0+A is the same representation in the next adjacent section and is stored in DIN B; and the quantity Q_0+A+B (which is the identical representation in the next buffer section) is stored in DIN. Similar action occurs in the second channel corresponding to buffer section DIN A', DIN B' and DIN' employing gate 165 and XFER 180 and XFER 182. If the condition YES (all zero word) occurs, then the action is identical, except the value input to DINA is the FALSE value for the word (CH 1) and the value input to DINA' is the FALSE value for the word (CH 2).

On the other hand, if the condition FAIL•NO occurs, then the following action is different, and

the particular cycle of forgiveness for the particular function is important. Firstly, in the condition FAIL*NO, the sensed data is not at all employed, rather the corresponding quantity from FALSE store 25 is gated onto bus 150. Furthermore, under these circumstances, and assuming 0 cycles of forgiveness, then gates 162–164 pass the data from bus 150. That quantity is directly stored in DIN (via gate 164). On the other hand, if the quantity had I cycle of forgiveness, then gate 164 is unenabled and the contents of DIN remain unchanged. Rather, the contents of DIN A are altered. Finally, if the quantity had II cycles of forgiveness, then gates 163 and 164 are disabled, thus quantities in DIN and DIN B are unchanged and the quantity from the false store coupled via bus 150 is used to only write DIN A. Similar action occurs in the other buffer section, e.g. buffer portions DIN A', DIN B' and DIN'.

As a result, and especially since registers DIN and DIN' contain quantities which will be immediately used in logic evaluation, there is no delay in rendering effective a false or corrupted value corresponding to functions with 0 cycles of forgiveness. There is a 1-cycle delay in rendering effective those quantities for functions with 1 cycle of forgiveness, and a two cycle delay in rendering effective those quantities respecting functions with 2 cycles of forgiveness.

Function Evaluation

The apparatus shown in Fig. 3 is useful in illustrating the function evaluation. The input data on which function evaluation occurs is located in the set of registers DIN, OCK, CS, LA and CR, for respectively direct input, output check, control store, latched expressions and current results. The manner in which information is circulated to the direct input buffer (DIN) has been explained above. A similar process (which will be described hereinafter) takes place using, as inputs, the state of the output functions. At least those of the output functions which are inputs for the purposes of evaluation are located in the OCK buffer. The control store (CS) is a buffer which includes non-vital data derived from an operator input keyboard or the like. Certain expressions in the Boolean expression list which must be evaluated may contain latched functions. Such a function is one in which at least one of the terms to maintain it true includes the presence of the output function itself. Latched expression data is found in the buffer LA. Finally, some functions important in evaluating the expressions are the result of the evaluations of previous functions in the same cycle. This data is found in the buffer CR.

The Boolean expression list contained at storage device 500 relates the condition of an output function to one or more input/output functions in the form of the sum of products such as the equation shown in Fig. 3 (i.e. $OF_1 = IF_1.IF_3 + IF_2.IF_4 + \ldots$). This equation represents that output function 1 (OF_1) is defined by the product of input function 1 (IF_1) and input function 3 (IF_3) or input function 2 (IF_2) and the inverse of input function 4 ($\overline{IF}_4$), etc. Using these terms as exemplary, if both input functions 1 and 3 are in their logic 1 condition, then output function 1 will be in its 1 condition. On the other hand, if input function 2 is in its 1 condition and input function 4 is not in that condition, then that is another condition which will render output function 1 in its 1 condition. The Boolean expression list 500 maintains a series of these definitions, and these are evaluated in turn. The Boolean expression list is addressed by a control counter 501; when any particular definition is addressed, the terms in the definition are sequentially accessed via the addressing device 502. For example, if the equation shown in Fig. 3 is addressed, the first function (input function 1) must be evaluated. Device 502 locates the present condition of the input function in one of the buffers. The present condition of the input function 1 is identified by its T/F bit (the least significant bit). This bit is output from the appropriate buffer on the line 503. Typically, that bit will pass unaltered to an AND gate 505. However, if the function is inverted (such as input function 4), then the bit will be inverted by inverter 504. AND gate 505, in the case of second or subsequent factors in a single term, AND's the evaluated bit with the contents of the T/F bit store 506; the first term is inserted directly in the T/F bit store 506. In this fashion, after the T/F bit in each function of a single term has been addressed, the T/F bit in each function of a single term has been addressed, the T/F bit store 506 determines whether the term is true or false; if true, then the output function is evaluated as true; if false, the next term in the definition must be evaluated in the same fashion. Accordingly, a zero result in the T/F bit store 506 calls for evaluation of the next term by stimulating control counter 501. On the other hand, a true output calls for evaluation of the next equation. However, preceding the evaluation of the next equation, a vital evaluation is made on the equation whose preliminary evaluation was just completed.

In general, the vital evaluation involves reevaluating the same term of the equation which made the output function true. However, in this instance, rather than evaluating merely the T/F bit only, the entire unit is "added" in a polynomial divider with all other values which are factors of the term. In our example, input functions 1 and 3 would be "added" in the polynomial divider. As indicated above, there are two channels of computation and the unique values of functions in each channel are in a different respective code set. Thus, the PD 509 is personalized prior to each computation for the appropriate code set. This personalization sets the state of the feedback network. Evaluation of channel 1 input expressions produces a result (also 32 bits

long) which is also in the first code set. Since each true value is different, and since it is a necessity that the output function evaluate to a single value (the name for the particular function) regardless of the particular term in the equation which makes the expression true, we must somehow compensate for the fact that different input functions must yield the same output function. This compensation is provided by a preconditioning store 507. The preconditioning store 507 has a preconditioning constant for each different term of each different equation. The preconditioning constant compensates for the different product terms so that regardless of which term is evaluated, the output value will be identical. Thus, vital evaluation of an output function includes preconditioning the polynomial divider with the appropriate preconditioning constant, loading in a value for the first factor in the term, and thereafter loading in a value for each other factor in the term. At the conclusion (for channel 1 computations) the result should be a code word in the first set. The polynomial divider is arranged to check that fact. If the code word is indeed one of the small number of code words in the first set, then it is treated as the correct result and RESULT in transferred to the X buffer 510. On the other hand, if the expression result is not in the appropriate code set, then the polynomial divider 509 initiates access to the false word store for this particular expression result and FS is stored in the X buffer 510 (the output buffer).

If, on the other hand, during the expression evaluation, all terms in the expression are used and no term is found which makes the function true, then the signal ATU is produced; this signal has two effects, e.g. it signals evaluation of the next expression in the Boolean expression list 500 and it also accesses the false store to produce FS for storage in the X buffer.

In this fashion, then, a list of Boolean expressions can be evaluated and the X buffer loaded with either the false value, or the true value. It is important to note that while the FALSE value is extracted from the FALSE STORE, the true value is computed, i.e. it is not stored anywhere. Since the correct true value is one of a small set, computation of the true value requires flawless processing. The test for the true value may not catch all errors, but the ones which are not detected are of low probability and the processing is designed to detect these errors later. This completes the description of channel 1 processing. We have in addition to the channel 1 data values, channel 2 data values and they are used as follows.

Rather than looking through each equation again to find a term which makes the equation true, channel 1 processing has identified a particular term in each equation (if the equation to be evaluated is true) which makes it true. In channel 2 processing, the processing can be shortcut somewhat; first the personalization of PD 509 is changed to the second code set, second, a preconditioning constant corresponding to this particular term (for channel 2) is accessed from the preconditioning store 507 to precondition the polynomial divider 509. Thereafter, and in turn, the channel 2 values for the functions in the term making the expression true are accessed from the appropriate buffer and loaded into the polynomial divider 509. At the conclusion of the operation of the polynomial divider, the expression result should be a code word in the second code word set. If it is, the RESULT is loaded in the X buffer 510 in an appropriate position; on the other hand, if an expression results which is not in the second set, then the false word is accessed from the store and the expression FS is loaded in the X buffer at the appropriate location. As we will see below, those expressions which are evaluated as valid in channel 1 but are found to be false in channel 2, will be hereafter treated as false. Only expressions in which true results were validly evaluated in both channel 1 and channel 2 will be treated as true.

Output and Output Check Processing

The output and output check processing begins with the results of the evaluation stage stored in the X buffer 510 and the X' buffer 510'. Two processes must be performed, i.e. first the output ports must be controlled to be either on or off, depending on the results which are reflected in the X and X' buffers. Furthermore, the actual state of the output ports must be checked against the allowed state (as reflected by the evaluation results) in order to determine that the point is or is not in the appropriate condition vis a vis the evaluation result. Note that the vital processor does not make the determination, the vital processor only prepares the appropriate check word so that the result can be determined elsewhere (in the VRD).

As shown in Fig. 6, the output function is effected from the contents of a V buffer 520. The V buffer is loaded, at the appropriate time, by merely examining an appropriate bit (the T/F bit, or least significant bit in the expression result) in the evaluation result stored in the X buffer, copying that bit to the V buffer in the appropriate location and, at the appropriate time controlling each output port so as to reflect the status of its corresponding bit in the V buffer 520. Whenever the contents of the V buffer 520 are copied to the output ports, an USEY flag is set, the reason for this action will appear shortly.

The recheck processing employs a more comprehensive complement of buffers. These include the Y(E) buffer 530 and the corresponding Y(O) buffer 540. Recheck operations are divided into even and odd 50 ms. cycles. The contents of the buffer 530 and 540, however, are written on each one second cycle in the following fashion. The corresponding positions in the X and X'

buffers are examined. If both positions indicate a "true" or on condition for the corresponding port, then particular words Y(E)(T) and Y(O)(T) are constructed (described below) and inserted into the corresponding location in the buffers 530 and 540; if both the values at the X and X' buffers do not reflect a "true" or on state for the port, then different values ('0000') are copied into the appropriate location in the buffers 530 and 540, respectively. Accordingly, whereas the condition of the X and X' buffers reflected only channel 1 or channel 2 processing, respectively, the contents of buffers 530 and 540 represent a combination of results in both channels, in a restrictive sense in that the true value is in the buffer 530 and 540 only if both channel 1 and channel 2 processing agree on a true value, otherwise the false value is found in buffers 530 or 540. As is described below, the Y(E) values are accessed on even recheck cycles and the Y(O) values are accessed on odd recheck cycles.

Since, as has been described above, the main cycle processing (which is used to set the output ports in accordance with the status of the V buffer 520) and the recheck cycle (the processing for which is now being discussed) are asynchronous with each other, we have to have some means of locating the appropriate data. More particularly, this recheck processing will produce check words which correlate the state of the output ports with the state that the output ports should have, as reflected by the internal data. Because of the asynchronous relation between the recheck processing and the main cycle processing, the internal data which should be reflected at the output ports may have already been overwritten in the buffers 530 and 540 by the time the recheck processing takes place. Thus, a lack of correspondence could be signalled only because the output ports have not yet been controlled in accordance with the data appearing in buffers 520, 530 and 540. In order to avoid this situation, a further set of buffers Y(E)N−1 and Y(O)N−1 are provided. These buffers are used to hold the results of earlier main cycle processing (from the previous cycle, i.e. cycle N−1) for comparison with the output ports if the output ports have not yet been controlled to reflect the condition of the buffers 520, 530 and 540. A flag, the USEY flag, is used to determine whether the data in buffers 520, 530 and 540 will be used for comparison or, in leiu of that data, the data in buffers 525, 535 and 545 will be used. When the output ports are controlled to reflect the data in buffers 520, 530 and 540, then the USEY flag is set. If, prior to recheck processing, the USEY flag is set, then the data from buffers 520, 530 and 540 is used. On the other hand, when the data from buffers 520, 530 and 540 is transferred to the buffers 525, 535 and 545, respectively, then the USEY flag is reset; if that flag is reset during the recheck processing, then the data from the latter set of buffers, i.e. 525, 535 and 545 is employed.

As has been referred to above, the recheck processing occurs in even and odd cycles and therefore even or odd data must be selected. These selections are effected by gates 550, 551 and 552, the former two gates are controlled by the USEY flag and the latter gate operates in dependence on whether or not the recheck cycle is even or odd. Accordingly, as shown in Fig. 6, at the appropriate time in the recheck cycle, the gates 550–552 are controlled to pass the appropriate even or odd data to the recheck test. The recheck test is functionally illustrated in Fig. 7.

As shown in Fig. 7, a ROM 600 has stored even and odd values TREE(T) and TREO(T). In the course of recheck processing, either the even or the odd values are placed on a bus 30' (similar to the bus 30 of Fig. 1A). Each output port has an absence of current detector (or AOCD) which is similar in purpose to the sense circuits (see Fig. 1H), discussed below). The bit patterns from the ROM 600 are applied to the bus and through the bus (and through a corresponding signature element similar to 35–1) are appled to the AOCD's associate with each port. In the even that the port is off or false, the AOCD repeats the input bit pattern in its inverted sense applied to it, if the output port is on (or true) the AOCD returns a null value. A transposition apparatus similar to the transposition apparatus 40 (of Fig. 1A) is interposed between the AOCD and the RAM buffer TRETMP 610. The transposed data from the AOCD's are stored in the buffer 610, necessarily in signature order, since that is the order in which they are returned. The reader will note a similarity between this output port check and the input port sensing of Fig. 1A. The raw data is written to TRETMP in signature order. Note that while ROM 600 stores TREE(T) and TREO(T), these are stored in transposed relation, relative to the presentation in TRETMP 610 to maintain the rule that the 'correct' result of vital values not be available to the machine.

At the conclusion of this processing, therefore, TRETMP has a value for each output port. If the port was on, then the value '000' is located in TRETMP at the location corresponding to the port. If the port was off, then the value TREE(T) or TREO(T) is stored in the corresponding location of TRETMP depending on whether the cycle is odd or even.

Once this processing is completed, the values from TRETMP are written into the buffer TRE 630. However, in this writing, the order is changed from signature order to logical order. Once the TRE buffer 630 is written, we now can effect the substantive recheck processing; a comparison of the actual state of the port (as reflected by the contents of TRE) with the state the port should be in based on the data passed through gata 552. It should again be emphasized that the vital processor does not actually make a decision on whether or not the port and

the data disagree, a check word is determined for each port or groups of ports, and the check word is passed to the vital driver (the other processor) where a decision is made as to whether or not a potentially unsafe failure has been detected.

Once TRETMP has been loaded, the recheck processing can calculate recheck checkwords. For ports allowed to be on, we have a 'true' word in Y(E) and Y(O), or in Y(E)N−1 and Y(O)N−1 (depending on recheck processing timing). For ports actually off, we have TRE(T) or TRO(T) in TRE buffer 630. In the absence of an error, the data should exist in either the Y buffer or the TRE buffer. In the recheck processing, we examine the Y buffer value, if the LSB is a 1, we use the Y value. If the LSB is a 0, then we use the TRE value, as shown in Fig. 7. As a result is a unique value which must be either:

YE(T) represents an "on" function which is allowed to be on, or TREE(T), represents an off function which is actually off.

As will be now decribed, the various values have been preselected so that these two values are in fact identical and thus form a tell-tale word whose presence is required for maintaining the effectiveness of the output from the vital processor.

The value YE(T) is constructed by combining a preconditioning constant PREYET with the true values X(T) and X'(T). The preconditioning constant PREYET is also arbitrarily selected. The value TREE(T) is the name selected for the 'TRUE' function TREE. Thus, given the selected values for X(T) and X'(T) as well as the selected value of TREE(T), the arbitrary value selected for the preconditioning constant PREYET is selected to make the two values noted above, in fact, identical.

Having derived, then, either YE(T) or TREE(T) it is now necessary to form a recheck check word to reflect this value so that after it is passed to the vital driver, a determination can be made as to whether or not the check word reflects faultless processing. While it is possible, theoretically, to use the selected value itself as the checkword, time constraints in one embodiment actually constructed, prohibit such a practice. Rather, in this embodiment a set of YE OR TREE values are collected to form a recheck checkword (where YE or TREE means the value YE or the value TREE). The number of such values in the set is in essence arbitrary, and is actually selected based on the amount of time available for execution of the recheck routine. In accordance with the embodiment actually constructed, the recheck checkword is generated by combining a number of the derived YE OR TREE values, as follows:

1. The polynomial divider is set to the selected 32-bit polynomial, e.g. the CH 1 32-bit polynomial, evaluate mode of the PD 509, as described hereinafter.

2. The polynomial divider is loaded with a selected preconditioning constant PREZE unique to a selected checkword to be formed.

Each of the YE OR TREE values are "added" into the polynomial divider in the order in which they were generated. The result in the polynomial divider is the checkword CHKZE.

This process is repeated for each set of values until a checkword is generated for each set.

The preceding description has been in terms of even cycle operation; odd cycle operation is identical except that rather than accessing YE values, 70 values are accessed; instead of extracting TREE values from buffer 630, values TREO are extracted; and finally the preconditioning constant is PREZO and the checkwords are identified as CHKZO.

Accordingly, each recheck cycle (20 of which are carried out interleaved with each main cycle) transfers a set of checkwords to the vital relay driver. This set of checkwords includes a pair of checkwords, CHKZE and CHKZO, for each set of results of the recheck test. However, in addition to these, the recheck check words also include other check words related, for example, to the clearing operation of the TRE buffer as well as the clearing operation of the TRETMP buffer. Finally, in each recheck cycle, a first recheck checkword is formed by the "sum" (via the polynomial divider) of all other recheck check words in that cycle.

The recheck check words CHKZE and CHKZO (as well as all other check and recheck check) words are processed through one more step before they are transferred to the VRD. We wish to ensure that check (or recheck) words from one cycle will not validate operations in any other cycle. The reason behind this desire should be apparent. To effect this goal, the primary processor alters each computed check (and recheck) word by a unique quantity. The VRD is arranged to effect a complementary process. As a result a check word CHKZE (for example) computed at the primary processor becomes CHKZE+U(n,k). U(n,k) is different for every check word on every recheck cycle (it is never repeated during the same one second main cycle). At the VRD, the received word CHKZE+U(n,k) is manipulated with U(n,k) to produce CHKZE. Since each processor (primary and VRD) independently determines U(n,k), the check words must be accessed by the VRD in the correct order. Any other order will not produce CHKZE (for example), and in the absence of CHKZE, the VRD will signal an error and prevent the effectiveness of the computed results.

Complete Description

Figs. 4 and 5 illustrate the input arrangement for the vital processor. Input actually has two

connotations, in a first connotation input refers to information input, which is the information which is processed in order to produce the output functions to operate various devices. However, input has another connotation in the sense of signals fed to the processor. In this connotation, input refers not only to input information, but output information as well. Therefore, in connection with Figs. 4 and 5 and the accompanying description, the "input" condition being sensed include not only input information but output information as well.

Fig. 4 illustrates a plurality of input sensing circuits 35–1–1 through 35–1–8, each arranged to sense one bit of input; in particular, whether or not the voltage between the terminals + and − is or is not in excess of a given threshold. The sensing circuits 35–1, etc. are shown in more detail in Fig. 1C and are further described below. Each sense circuit includes a drive input, as well as the condition sensing input (+, −). The drive input is derived from a latch 34 which has an output for each sense circuit. The input to the latch 34 is derived from the bus 30 and is loaded on the presence of the signal "load drive latch". This control signal is derived from the selector 32 which, as is illustrated, is capable of selecting any one of a LDL1 through LDL4 to produce the "load drive latch". The bit pattern latched at the time of "load drive latch" is determined by the condition of the different conductors in the bus 30.

Each of the sense circuits also includes a sense output terminal, and the different sense output terminals are coupled to different inputs of a signature element 35–1. As illustrated in Fig. 4, corresponding inputs and outputs of the signature element 35–1 are not necessarily connected together. This provides for a group signature as already described. The sense outputs, as modified by the signature element 35–1 are then applied to a data buffer 36. The buffer 36 operates in the presence of the "sense vital input" control signal which is derived from selector 37. As illustrated in Fig. 4, selector 37 can produce the "sense vital input" which is employed in the buffer 36 from any one of four inputs, SV1A through SV1D. When enabled, the sensed signals are applied to different conductors of the bus 30. Accordingly, it should be apparent that the arrangement of Fig. 4 meets the requirements in the description heretofore given.

Fig. 5 illustrates the remainder of the interface between the input and the processing device mainly for the purpose of transposing the sensed information as shown in Figs. 1A and 1B. More particularly, a plurality of shift registers 40–0 through 40–7 are connected to a series of conductors DB0 through DB7 which are in turn connected to corresponding conductors in the bus 30. Indeed, each of the shift registers has two sets of connections to these conductors. Each shift register has a serial input terminal connected to a different one of the conductors, e.g. shift register 40–0 has a serial input terminal connected to DB0, SR1 has a serial input connected to DB1...and SR7 has a serial input connected to DB7. Each time a different sensing bit pattern is applied to the sense circuits via the latch 34, and then enabled back onto the bus via buffer 36, the shift registers 40–0 to 40–7 have their contents shifted one place. After a number of sensing bit patterns (units) have been applied to the bus 30 in unit serial order equal in length to the length of the shift registers, each of the shift registers is full. In order to transpose the data so far described, the shift registers are read out in parallel. For this purpose, the shift registers can be read in turn via the demultiplexer 41. On reading a particular shift register, its contents are placed broadside or parallel on the conductors DB0-DB7. Those skilled in the art will understand that the eight conductors shown for broadside reading the shift register is only exemplary. The output of each shift register is placed on a different conductor in the CPU bus 30'. The length of the shift registers should match the number of conductors in the bus 30'. If the conductor 30' has 8 conductors, then 32 bits can be passed in transposed form of four groups of 8 bits each. Obviously, other arrangements are also possible.

Figs. 1A and 4 illustrate the use of the input sense circuits 35–1–1 through 35–1–n. A schematic of a suitable circuit is shown in Fig. 1C. The terminals IN+ and IN− are the input terminals to the control system, and the condition of these terminals (the voltage across them) is the parameter being sensed. An input which is on, has a positive voltage across the terminals exceeding a given threshold, and an input that is off has no voltage across the terminals or a voltage below the threshold. The driving bit pattern is applied to the input terminal DRIVE, and the corresponding output is taken from the terminal SENSE. In the absence of a voltage across the input terminals, the optical coupler LC3 is not transmitting, transistor Q6 is on and the voltage at the terminal SENSE, is low. This condition is maintained regardless of the bit pattern at the terminal DRIVE, i.e. when the input terminal is off, the bit pattern available at the terminal SENSE is a null bit pattern.

On the other hand, when there is a voltage across the terminals IN+ and IN− in excess of a given threshold, then the optical coupler LC3 can transmit, enabling the voltage at the terminal SENSE to go high. However, this condition is inhibited when the potential at the terminal DRIVE goes low. When the input at DRIVE is low, the optical conductor LC2 conducts, which inhibits conduction of LC3, allowing the potential at the terminal SENSE to go high. Only when the potential at DRIVE is high, will the output at SENSE (in the presence of an on condition at the input) be low. Accordingly, when the input terminal is off, the terminal SENSE is low, and when the input terminal is on, the voltage at SENSE is the inverse of the voltage at DRIVE. The sense

circuit 35–1–1 is vital in that no failure condition will allow the terminal SENSE to repeat, in inverted form, the big pattern DRIVE, and thus this pattern is indicative of an on terminal. Any failure in the sense circuit 35–1–1 will tend to simulate an off input condition, which, since it is restrictive, is a safe failure. While the circuit of Fig. 1C is that used in an embodiment of the invention which has actually been constructed, those skilled in the art will perceive how various changes and modifications can be made without departing from the scope of the invention.

In addition to sensing the condition of the input terminals, it is also necessary for apparatus to sense the condition of output terminals. Whereas in the case of the input terminal, any failure which simulated the presence of an on input condition (in the absence of such a condition) had to be guarded against, the converse is true in the output terminal. More particularly, sensing the output condition in a vital fashion requires that no failure mechanism be allowed which would simulate an off output condition when the output terminal is actually on. This is implemented, in the circuit shown in Fig. 1H, which is an absence of current detector (AOCD).

Referring now to Fig. 1H, a terminal ON/OFF is provided for controlling the condition of the illustrated, representative, output terminal. This terminal is driven by the output of the primary processor 3. As shown in Fig. 1H, the transistor Q2 is rendered conductive when the output is to be on, and at other times it is rendered non-conductive. When transistor Q2 conducts, the optical coupler LC1 transmits optically to enable the transistor Q1. Enabling transistor Q1 enables transistor Q4, which provides a current path through the winding W1 to the terminals OUT+ and OUT−. The diode D2, a light emitting diode, performs a telltale indicating, by the presence of an optical output, that the output terminals OUT+ and OUT− are in an on condition.

The remaining elements in Fig. 1H comprise the elements for sensing the condition of the output terminals OUT+, OUT−. More particularly, these elements include an additional pair of windings W2, W3, magnetically coupled by a core CR. So long as current is traversing winding W1, in excess of a given threshold, the winding CR is saturated; only in the absence of current (in which event the output is in an off condition) is core CR unsaturated.

For the purposes of driving the sensing circuit, a terminal DRIVE is coupled through a resistor to the base of a transistor Q3. Accordingly, in the presence of a positive voltage, the transistor Q3 conducts, and current flows through the winding W2. A voltage across the winding W3 is arranged to enable the transistor Q5, the collector of which is coupled to the output terminal SENSE. The windings W2 and W3 are arranged so that in the absence of a saturation condition of the core CR, the transition, in turning on the transistor Q3, will produce a voltage to enable the transistor Q5. Thus, a positive transition at the drive terminal produces a negative transition at the sense terminal, i.e. the driving bit pattern is inverted. However, this in only true when the core CR is unsaturated, for when the core CR is saturated, the flux produced as a result of the transition in the current in winding W2 does not produce a corresponding voltage in the winding W3, and therefore the terminal SENSE produces a null bit pattern. Furthermore, the circuit of Fig. 1H is designed so that there is no failure mechanism which would allow the driving bit pattern to be reproduced, in its inverted form, at the SENSE terminal, unless the output is in an off condition.

This operation is illustrated with the representative waveforms shown in Fig. 1J, including the waveform at the ON/OFF terminal, a waveform for the DRIVE Terminal, and the resulting waveform at the SENSE terminal. As shown in Fig. 1J, between times T_1 and T_2, the ON/OFF terminal is on. During this period of time, the terminal SENSE remains high regardless of the driving bit pattern because the core CR is saturated, and therefore transitions in the current level in winding W2 do not produce a resulting voltage in the winding W3.

However, at the time T_3, when the ON/OFF terminal is off, and the driving bit pattern is high, the sense voltage is low. At times T_5 and T_6, when the driving bit pattern is low, the sense voltage is high, and finally at time T_7, the driving bit pattern is high, the sense voltage is low. Thus, when the output terminals OUT+ and OUT− are on, the sense voltage does not respond to the driving bit pattern, but when the output terminals OUT+ and OUT− are off, then the bit pattern at SENSE is an inverted replica of the bit pattern applied at DRIVE.

Fig. 8 illustrates a block diagram of the relationship between the primary processor 3 and the polynomial divider 509. The polynomial divider 509 is described in more detail in Figs. 9 and 10, it suffices here to note that the polynomial divider 509 includes a 32-stage shift register with a controllable feedback network that can be personalized to operate in two different modes. A first mode can be termed the evaluation mode in which 32-bit quantities are manipulated. In this mode, the feedback network for the shift register can be personalized to one of two different code sets corresponding to channel 1 or channel 2. In this mode, the shift register will, in response to appropriate control commands, load a 32-bit quantity from the data bus, execute a controlled number of shifts, "add" modulo 2, a 32-bit quantity contained on the data bus, with the quantity contained in the shift register, and after a sequence of these operations, output a 32-bit quantity to the data bus.

In a second mode of operation, the shift register is loaded with a 14-bit quantity and, in

response to appropriate commands, produces an 18-bit quantity corresponding to the 18 check bits which are related to the 14 bits originally loaded, by one of two code rules, corresponding to channel 1 or channel 2 operations. In this so-called check mode, 14 bits are loaded from the data bus in the 14 low bit positions of the 32-bit shift register, and then with the shift register feedback network personalized to either channel 1 or channel 2 code rules, the shift register is shifted a predetermined number of times and produces an 18-bit result, which is the check bits associated with the 14 bits originally loaded, by the selected code rule. This latter mode of operation is useful to check whether or not a 32-bit word is or is not a member of the appropriate code set. This is accomplished by personalizing the shift register for check mode along with the appropriate code rule, loading the low order 14 bits of the word to be checked into the low order 14 bits of the shift register, and operating the shift register in the check mode to produce 18 check bits. These 18 check bits can then be compared, bit for bit, with the 18 high order bits of the 32-bit word that was originally submitted for testing. If the comparison is an equality, then the originally submitted 32-bit word is a word in the appropriate code set, and vice versa. Accordingly, the polynomial divider, and more specifically the shift register located therein, responds to the control commands shift, enable load, disable output, personalize channel 1 evaluate mode, personalize channel 2 evaluate mode, personalize channel 1 check mode and personalize channel 2 check mode. The specific control commands are provided by the primary processor 3, in a manner that will be apparent to those skilled in the art after reviewing the software description, below.

Fig. 9 shows the shift register (comprising stages Q_0–Q_{31} of the polynomial divider 509). Each stage has an input from an exclusive OR gate designated X, with a numerical subscript identical to the numerical subscript of the stage, and therefore exclusive OR gates X_0–X_{31} are present. In general, an exclusive OR gate can have up to three input, an input from the output of the preceding stage, an input from a gated input amplifier with the corresponding subscript or an input from a gated feedback buffer amplifier of corresponding superscript (i.e., X^1 through X^{31}). Gate G_0 is different in that firstly it has no input from a preceding stage, and secondly there is no feedback amplifier, the feedback path to gate G is always present. For enabling the loading or adding of data, the enable (E) control terminal gates the gated amplifiers A_0–A_{31} to provide the corresponding bit from a data bus to the associated exclusive OR gate. To personalize the feedback of the shift register, two control terminals are present, PE1 and PE2, the former personalizing the shift register for evaluation in channel 1, the latter personalizing the shift register for evaluation in channel 2. As can be seen from Fig. 9, the personalization corresponds to enabling different sets of the gated feedback buffer amplifiers, X^1, X^2 and X^{22} for channel 1, and X^{10}, X^{30} and X^{31} for channel 2. Another control terminal is the S or shift terminal which is coupled to the clock input of each of the stages Q_0 through Q_{31}. A positive going transition at the S terminal provides for a right shift, one stage per positive transition. Finally, the D control terminal provides for disabling the tri-state Q outputs of the stages Q_0–Q_{31}. Normally, that is when not disabled, the shift register outputs are available to the data bus.

Expressions evaluation uses the mode shown in Fig. 9 and performs the following operations:

1. The preconditioning constant is loaded into the 32-bit shift register, by placing the preconditioning constant on the data bus and enabling the E control terminal. The S terminal undergoes a positive transition and the data is loaded, e.g. the bit pattern at D_0 through D_{31} then appears at Q_0–Q_{31} (the D terminal is rendered high to disable the outputs and no feedback path is enabled).

Thereafter, the shift register is personalized for channel 1 or channel 2 operations by gating either PE1 or PE2. The S terminal is then pulsed a number of times, producing a right shift for each pulse, as indicated above, in an embodiment of the invention actually constructed, nine shifts are used.

To add a 32-bit quantity then, the 32-bit word to be added is placed in the data bus and thus appears at D_0 through D_{31}. The E terminal is gated and the S terminal is pulsed (and this time the D terminal is not gated) but the feedback is disabled. After this first shift, at any stage Q_n, the condition is the modulo 2 sum of D_n and Q_{n-1}, in the special case of Q_0, that stage is the modulo 2 sum of D_0 and Q_{31}.

After the modulo 2 addition, the S control terminal is pulsed n times (again n is 9 in the embodiment constructed). This last sequence of steps is repeated for each factor to be "added". The result in the shift register can then be parallel loaded to the data bus for use by the primary processor 3.

Fig. 10 is a detailed schematic of the shift register in the polynomial divider 509 in the check mode. Comparing Figs. 9 and 10, the differences between the evaluate mode and the check mode should be apparent. For one thing, whereas in Fig. 9 (evaluate) there is a feedback path from the 32nd stage (Q_{31}) to the first stage (Q_0), in the check mode, that feedback path is broken. Rather, there is a feedback path from the 14th stage (Q_{13}) to the first stage (Q_0), and there is no other feedback in the first 14 stages. The feedback path from the 32nd stage (Q_{31}) is taken back to the 15th stage (Q_{14}). There are two further feedback paths which are condi-

tional on the particular channel being operated on, e.g. the 22nd stage (Q_{21}) for first channel operation and the 26th stage (Q_{25}) for the second channel operation. First channel operation is personalized by gating the terminal PC1, to enable the gated buffer X^{21}, and second channel operation is personalized by gating the terminal PC2 to enable the gated buffer X^{25}. The switching arrangement (under control of the primary processor 3) to personalize to the evaluate or check mode, although not illustrated, should be apparent to those skilled in the art. Furthermore, the gated buffer amplifiers which are not involved in the evaluate mode operation are not shown in Fig. 9, and similarly, the gated buffer amplifiers not involved in the check mode are not shown in Fig. 10, for clarity.

In order to operate in the check mode, the primary processor, by controlling the polynomial divider 509, performs the following procedures.

A 14-bit information field (that is, the T/F bit and the following 13-bit "name") are placed in the lower order 14 bits of the data bus, and the E terminal is gated to enable the gated buffer amplifiers A_0 through A_{13}. At the same time, the shift terminal S is gated to clock each stage of the shift register. After this operation, the state of the shift register is equivalent to the corresponding data bus bit ($Q_n=D_n$, for n=1, 2, 3, ... 13). And Q_n is 0 for n=14, 15, ...31.

Thereafter, the terminal S is pulsed, 14 times, producing a right shift of the contents of the shift register for each pulse at the terminal S. If we are operating in channel 1, the terminal PC1 is also enabled or if we are operating in channel 2, terminal PC2 is enabled. The former enabling the gated buffer amplifier X^{21}, the latter enabling the gated buffer amplifier X^{25}. At the conclusion of the operation, the states of the 32 stages of the shift register Q_0 through Q_{31} are in the following condition, the original 14-bit information field now resides in its original location, e.g. Q_0–Q_{13}, but now instead of the 18 bits Q_{14} through Q_{31} being 0, they correspond to the check bits in the appropriate code set (either channel 1 or channel 2) corresponding to the 14-bit information field.

The 14-bit information field which was originally loaded (D_0–D_{13}) was derived from a 32-bit value calculated or sensed in accordance with the preceding description. The purpose for loading the 14-bit information field was to derive the appropriate 18-bit check field in order to compare the 18-bit check field so derived with the 18-bit check field originally associated with the 14 bits in the primary processor 3. With the derived 18-bit check field (in stages Q_{14} through Q_{31}), we can now compare these 18 bits with the 18 bits that had been associated with the 14-bit information field in the primary processor 3. Any suitable operation can be used for this comparison (for example an exclusive OR), and if the comparison is not an equality, then we have determined that the 32-bit word was not a word satisfying the appropriate code rules.

Software Overview

System software is organized into a number of different sections:

1. Sense input state of vital inputs.
2. Evaluate Boolean expression list.
3. Set vital direct outputs to states corresponding to the appropriate evaluated expression.
4. Accumulate main cycle check words on this cycle to be sent to the vital relay driver at the beginning of the next main cycle.
5. Perform vital "recheck" of vital outputs every 50 ms and send recheck check words set to the vital relay driver every 50 ms.

Each "entity" internal to the system is assigned 2 pairs of names. Each name is a unique 32-bit word. This 32-bit word has 2 fields, an information field and a check field. The information field is 14 bits long, so that there are 16K such names. These names are arranged in pairs, each pair representing a "true" value and a "false" value. The pair is determined by assigning a 14-bit word whose least signficant bit is a "0" as the "false" value of the pair, and assigning the complemented 14-bit word as the "true" value of the pair. Thus, there are 8K such pairs, or, there are 8K entities which can be assigned a unique name paid.

The 18-bit check field is then determined by polynomial division using a certain code.

A name pair is associated with each entity, and the check field for the name pair is determined by code C1, 1. (C1, 1 is merely the feedback configuration which will give us the 18-bit check field if we load the 14-bit information ield into the PD 509 and operate it in the check mode, as described above.) This name pair is associated with the "channel 1" name of the entity.

Another code, C1.2 is then used to form different check fields for the same name pair information fields. This name pair is associated with the same entity and is called the "channel 2" name of the entity. Again, C1, 2 merely identifies the feedback configuration of PD509 which will produce the 18-bit check field if we load the 14-bit information field and operate the PD509 in the check mode, as described above.

Now it can be seen that there are 2 pairs of names associted with each entity, one pair for CH1, the other for CH2. The "false" value of each pair has the same information field, but a different 18-bit check field. The "true" value of each pair has the same information field and this

field is the complement of the "false" name field, and again the 2 18-bit fields are different.

The reason that the names are assigned as "codewords" is that a non-vital test can be made to see if any entity name is valid without actually knowing what the value of the name should be. This non-vital test is made as follows:

If for instance a 32-bit "true" value is circulated (actually the complement of the "true" value is input so that the 'true' value may be returned) through the vital input circuitry of an input port to determine the port state, the returned value may be the actual "true" value (if the port was "on"), or it may be "0000" (if the port was "off") or it may be a corrupted value.

The returned value can be checked for validity by putting its information field into the polynomial divider or PD (using the appropriate code configuration) and shifting the PD the prescribed number of times. If the resulting check field thus generated matches that of the returned 32-bit value, the returned value is a codeword. Since there are "2 to the 32" (approx. 4 billion) 32-bit words, and there is only a small set of "2 to the 14" (approx. 16 thousand) codewords, the probability that the returned value was corrupted into another code word is 1 in "2 to the 18" (approx. 1 in 250 thousand).

It must be emphasized that this is only a non-vital check, used to keep "corrupted" data out of the system. If an input parameter was corrupted into another code word, however, it would not evaluate correctly in the expression evaluation anyway.

A Note About "Order"

In the following descriptions of the system software, terms like "logical order", "signature order" and "physical order" will appear in reference to the order in which values representing information about input and output ports are listed within a buffer. These terms are defined as:

Physical Order

This is the order in which the input or output port circuits appear on the printed circuit board. It has little or no significance to the software, but is used in assigning input and output functions to particular circuits on the board. Once the physical functions have been assigned to the board circuits, the "logical order" is determined.

Logical Order

This term can best be understood by considering an example. In an embodiment which has been constructed, there are 16 input ports on each input board. There are 16 data bits in the data bus. Each bit in the data bus (bit 0 to bit F–hexadecimal notation) is assigned to the "permanent side" of the input port sense circuit. Thus, a list of the input functions assigned to one input port board, if listed in the order which is determined by the data bus bit connected to the "permanent side" of the input circuits (starting with bit 0 and ending with bit F), would be in "logical order".

Therefore when the processor outputs a set of 16 32-bit words to circulate through the input port circuitry (1 32-bit word circulates through each of the 16 input port circuits), the word meant for "logical" input port number 0 is transmitted on data bus bit 0. The same goes for port numbers 1, 2, 3, ..., F.

Whatever goes out has to came back, and the transmitted data is eventually returned for reading by the processor. The order in which the words are returned, however, is not logical order at all. It is in "signature order".

Signature Order

The "non-permanent" side of the input port circuit is the "data sense" line. This line ends in a programming pin on the input board and is then jumpered to another programming pin which connects the "data sense" line to one of the 16 bits on the data bus. The pattern in which all 16 of the "data sense" lines of the 16 ports in group X, Y are connected through programming pins to the data bus bits is called the "group signature" of the input port group.

There are 16 such group signatures defined, and one of this predefined set must be used.

The group signatures may be defined as follows:

If "AIN" is the data bus bit connected to the permanent side of the input circuit and if "AOUT" is the data bus bit connected to the "data sense" side of the input circuit, then a list showing AIN from 0 to F corresponding to the AOUT bit it is connected to defines the group signature.

Next, there is the concept of "supergroups". There may be a maximum of 16 input boards in a supergroup. Each supergroup has an I/O interface board connecting the set of up to 16 input boards (input groups) to the CPU. Each of the input boards within one supergroup must have a unique group signature.

The I/O interface board also has a signature associated with it. This is called the "supergroup signature", and is defined as follows:

The I/O interface board has 16 8-bit shift registers. The input to each serial shift register is

connected to a programming pin. This pin is in turn connected to one of the 16 bits of the data bus. The pattern in which the shift register inputs are connected to the data bus bits on the I/O interface board is called the supergroup signature. Each supergroup signature in the system is unique. There is a set of 16 predefined supergroup signatures, and each signature must be one of this predefined set.

The shift registers on the I/O interface board are accessed in a fixed order, i.e. SRO (shift register O) is read by the processor by addressing the I/O interface board with address bits A4 through A1=0. This address will always address SRO regardless of the supergroup signature pattern, however, the contents of SRO vary with the signature pattern.

Suppose a 32-bit parameter AI1 (O,X,Y) (T) is circulated through input port O,X,Y. (O,X,Y means logical port 0 in group X, supergroup Y.) Which SR on the I/O interface board will this data end up in? Assuming that the group signature for group X is such that AIN(O) corresponds to AOUT(7), and that the supergroup signature 7 connects data bus bit 7 to the input of SR5. Then data AI1 (O,X,Y) (T) will end up in SR5 as a result of the two signature transpositions, the group X signature and the supergroup Y signature.

The order of the returned input port parameters within the section of the DIN buffer corresponding to group X, Y will be that which is determined by the effect of the two signatures.

All this being said, it now follows that the order in which the values of the input port parameters in DIN is that determined by the effect of the two signatures effecting input port group X, Y. This is called "signature order".

The reason that signatures are used at all is as follows.

Through the other techniques described above, it is ensured that when a bit pattern is calculated through a sensing circuit, it will be returned in its inverted form if and only if that sensing circuit is in a logical 'on' condition. Two different tests are relied on to ensure that no errors have crept into this sensing function. The first test is the non-vital test to assure ourselves that the sensed word is a word which satisfies the code rules set up in advance, i.e. does the 18-bit check field correspond to the 14-bit information field in accordance with the appropriate code set? The vital test generates a check word which will not be the expected check word unless every bit of the 32-bit sense word is exactly as expected. If the check word so computed is not the expected check word, then the vital driver will detect this and not allow the results to become effective. However, consider the following: a 32-bit sense word is circulated to a particular input port (in particular, the one which is associated with the 32-bit word). However, because of a malfunction, the 32-bit word is actually also connected to another port. Assuming that the "right" port is off, and the "wrong" port is on, and that the "wrong" port is in the same physical position as the "right" port, but on a different board, then these circumstances, the conductor on the data bus associated with both ports will "see" a null pattern from the "right" port, and the inverted pattern from the "wrong" port. Thus, the response from the "wrong" port will be placed on the data bus and will be accepted by the remaining components of the processor and stored in the location for the "right" port. This would give the appearance that the "right" port is on, when in actuality it is off. This is an unsafe failure. The scrambling effected by the signature order, different for different groups, means that the response from the "wrong" port will not be directed to the same data conductor as is the response from the "right" port. This will ensure that this error is detected.

Input States of Vital Direct Input Ports

The states of the vital direct inputs are determined by circulating two 32-bit parameter values through the vital input circuitry of each input port (one 32-bit value for each of two channels).

The returned value from this circulating operation in non-vitally checked for integrity by passing it through the PD (polynomial divider) to derive its check bits. If the check bits derived match those contained in the returned value, then it is a "codeword", and is assumed to be the "true" value, (indication that the input port is "on"). If the value returned is not a codeword, the "false" value is substituted in its place.

This operation is carried out for CH2 values also. Thus, the representation of each input port value is a total of 64-bits. These values are re-generated each main cycle (1 sec.).

Evaluate Boolean Expressions.

An important feature of the software system is the list of Boolean expressions which define the logic of the interlocking. This list of expressions must be "primordially" safe, that is the expression list, and the order in which they are executed, must, when executed accurately, operate the signals, switches and other vital hardware at the interlocking to allow the safe passage of trains through the interlocking.

The expressions are in "sum-of-products" form and arranged in order to execution. Each expression can contain any number of product terms, which in turn can contain any number of functions.

Each expression is evaluated in two channels, and each expression in each channel produces a

32-bit result which, if correct, is a codeword, particularly in C1, 1 (CH1), or C1, 2 (CH2).

There are several different types of expressions:

A. Expressions whose result is used to determine the state of a vital output (a direct output port, for example).

B. An expression whose result is used as a function in a subsequent expression in the list. The result thus generated is called a "current result", since it is valid only for the current main cycle.

C. An expression which contains its own result as a parameter in one or more of its product terms. This is called a "self-latched" expression, since it is the equivalent of a latching relay (a relay which sustains power to its coil through its own front contact).

After the expressions have been evaluated, those expressions which determine the states of the vital direct output ports are non-vitally sampled (actually only the CH1 expression result list is sampled), and the corresponding output ports are set to "on" or "off", depending on the expression result.

In addition to setting the output ports to their "assumed" correct state, two buffers of 32-bit data are compiled which represent a combination of the CH1 and CH2 expression results. These buffer are used by the "vital recheck" routine to assure that the states of the direct output ports are in their permissive states only if the expression results of both channels (64-bits) allow that direct output port to be "on". This vital assurance is left to the "recheck" routine which executes every 50 ms.

Transmit Main Cycle Checkwords to the VRD

The final arbiter of safe operation in the system is the VRD (vital relay driver). The main software must prove that it has performed all of its vital main cycle operations correctly in order to satisfy the requirements of the VRD which in turn uses this information to keep the vital relay energized. The vital relay allows power to be delivered to the direct output ports of the system through its front contacts. Each bit in the set of main checkword data must be correct for the VRD to generate a modulated digital output of a certain frequency which energizes the detector 5 so as to energize the coil of the vital relay.

The set of main checkwords is accumulated throughout the current main cycle and delivered to the VRD at the beginning of the next main cycle.

The main checkwords assure that all the internal buffers used by the system software have been vitally cleared, so that data operated on during the current main cycle has been generated during that cycle, i.e. it is not "old" data.

Vital Direct Output "Recheck"

The state of each vital direct output port is checked every 50 ms by circulating a 32-bit parameter value through the A.O.C.D. (absence of current detector) of the output port.

If the "true" value which was circulated through the A.O.C.D. is returned inverted, it provides assurance that the output port is in its "off" state. Only those output ports whose corresponding expression results (in both channels) are "true" are allowed to return any value other than the "true" value circulated through the A.O.C.D.

The recheck cycle checkword set is a set of 32-bit checkwords which represent the correspondence of expression result values with the actual states of the direct output ports.

The recheck cycle uses different data every other cycle. On the "even" recheck cycle, CH1 type values are circulated through the output A.O.C.D.'s, while on the "odd" recheck cycle, CH2 values are circulated. This provides "32-bit protection" over each 50 ms recheck cycle, and "64-bit protection" over 2 recheck cycles.

CLAIMS

1. A method of computing a multi-bit binary value of significance from two or more input multi-bit binary values comprising:–

a) providing a feedback shift register, with plural stages, arranged to provide maximal length sequences,

b) preconditioning said feedback shift register by controlling each stage thereof to attain a condition identical to a corresponding bit of a predetermined bit pattern,

c) exclusive OR'ing one of said multi-bit binary values, bit by bit, with a corresponding stage of said shift register, and

d) repeating said step (c) for each other of said input multi-bit binary values to produce a resulting bit pattern in said feedback shift register, to achieve at the conclusion of said step (d), said multi-bit binary value as equal to the resulting bit pattern in said feedback shift register.

2. A method according to Claim 1 in which said step (b) includes the step of shifting said feedback shift register a given number of times after each of said stages is controlled.

3. A method according to Claim 1 in which said step (c) includes the step of shifting said feedback shift register a given number of times after said input multi-bit binary values is

exclusively OR'ed into said feedback shift register.

4. A method according to Claim 1 in which said step (b) includes the step of shifting the feedback shift register a first given number of times after each of said stages is controlled and, in which said step (c) includes the step of shifting said feedback shift register a second given number of times after said one of said input multi-bit binary values is exclusively OR'ed into said feedback shift register.

5. A method according to Claim 4 in which said first and second given number are equal.

6. A method according to any preceding Claim in which each of said input multi-bit binary values and said predetermined bit pattern have a bit length equal to the number of stages of said feedback shift register.

7. A method according to any preceding Claim which includes the further step of testing said resulting bit pattern to cofirm it includes two fields, a first information field, and a second check field related to said first information field by a predetermined code rule.

8. A method according to Claim 7 in which each of said input binary values also include a first information field and second check field, which second check field is related to the corresponding first information field by the identical code rule.

9. A method of computing a multi-bit binary value substantially as described hereinabove with reference to Figs. 9 and 10 of the accompanying drawings.

Printed in the United Kingdom for Her Majesty's Stationery Office, Dd 8818935, 1986, 4235.
Published at The Patent Office, 25 Southampton Buildings, London, WC2A 1AY, from which copies may be obtained.